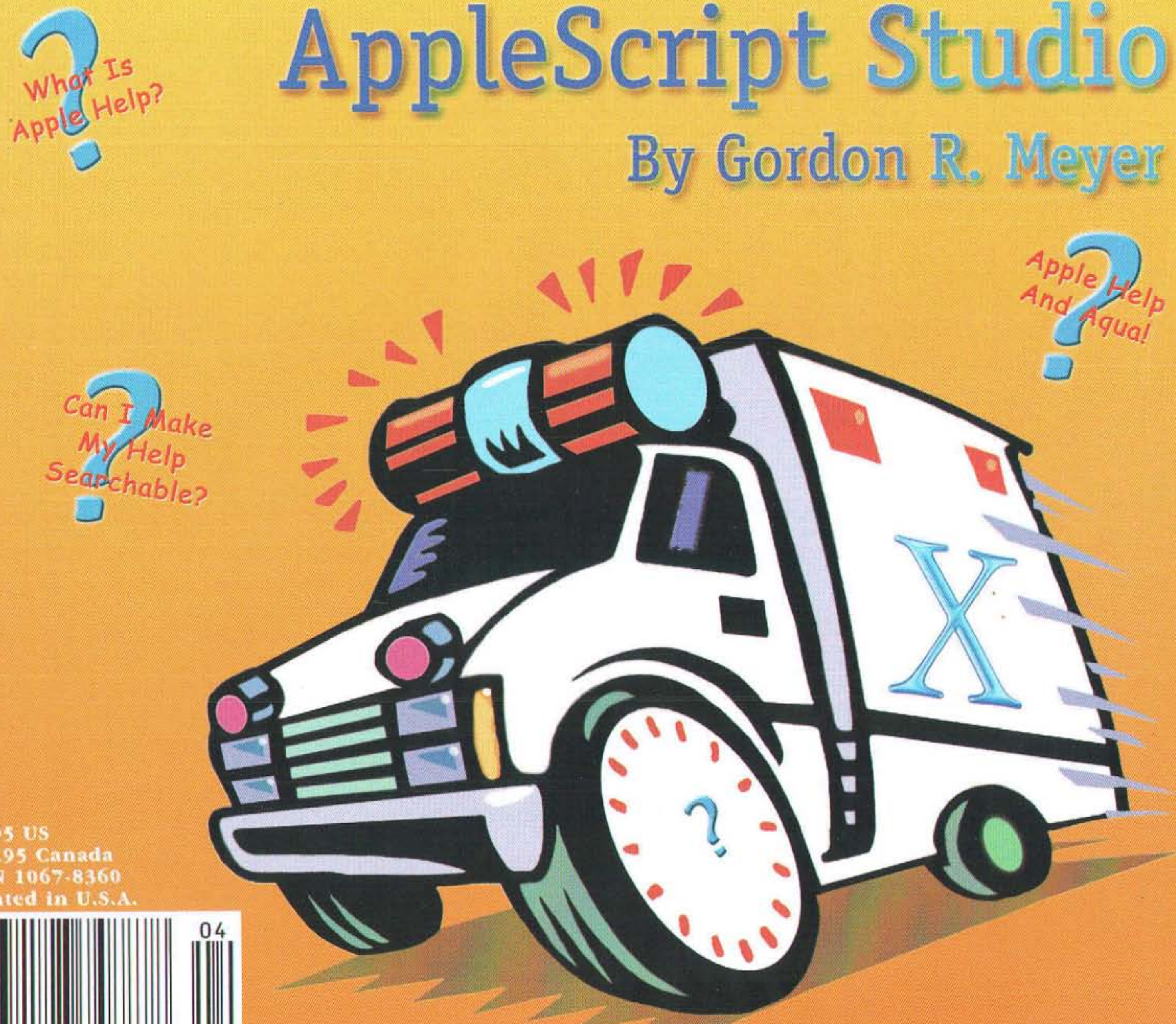
MacTech Magazine
Vol. 18, No. 04 • April 2002

MacTech®

The Journal of Macintosh Technology and Development

Apple Help & AppleScript Studio

By Gordon R. Meyer



$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.

HALF THE CODE*

- **Write once**, run on any platform
- **Database** access & **Internet** protocols
- **Client-server** • **Multimedia**
- **CGI** processing • Rapid **prototyping**
- Blisteringly **fast** • **Easy** to use
- **English-like** programming language

- Develop standalone applications for:
  Mac OS 9 | OS X | Windows | Linux

• **Free trial**	**www.runrev.com**
• **Email**	**info@runrev.com**
• **Phone**	**+44 131 718 4333**

"THE HOLY GRAIL OF CROSS-PLATFORM SOFTWARE DEVELOPMENT"

MacUser UK - 5 mice - Nov, 2001

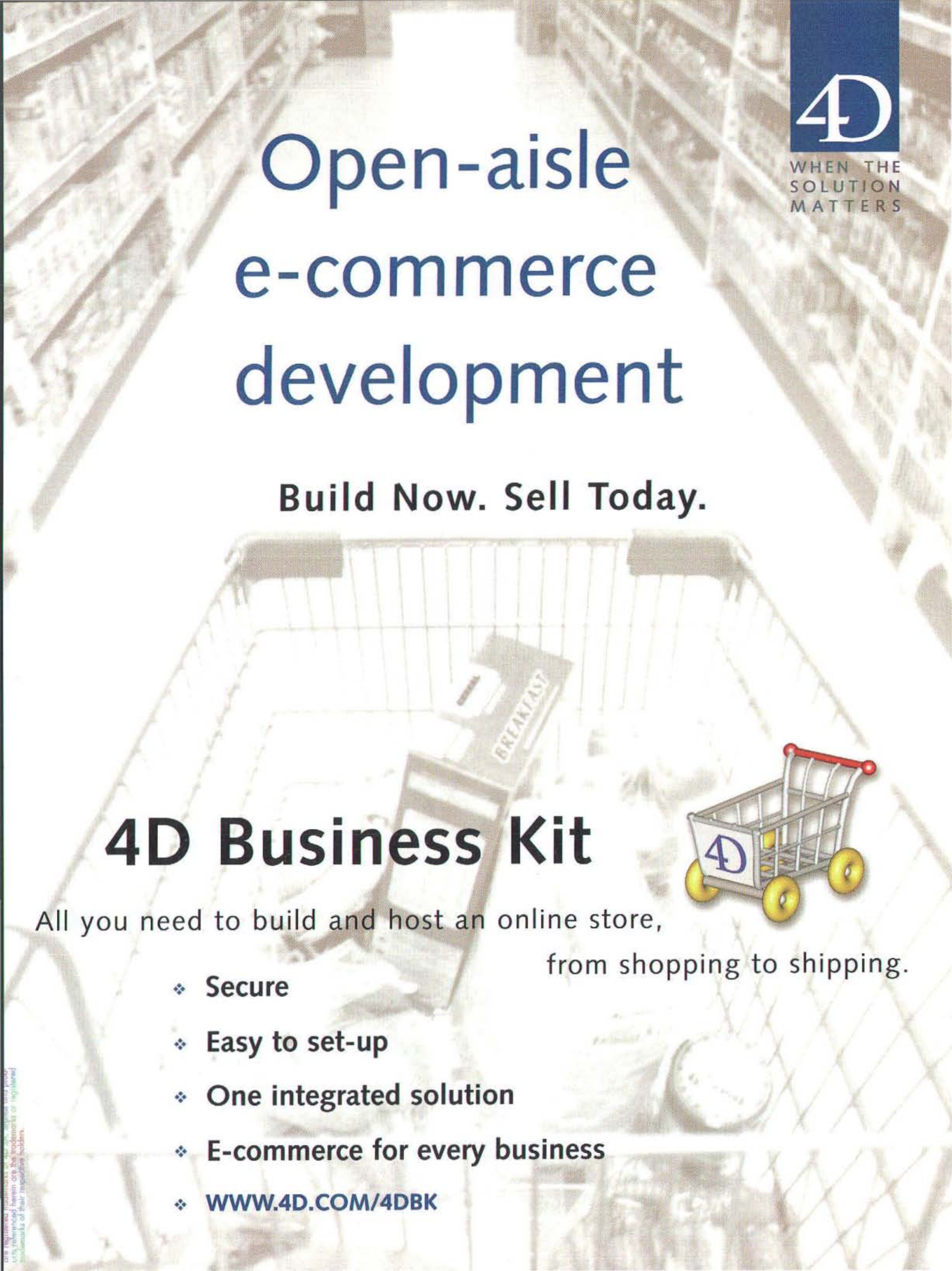
REVOLUTION 1.1

Power to the Developer!

RUNTIME REVOLUTION

Runtime Revolution Ltd, 91 Hanover Street, Edinburgh, EH2 1DJ, UK

* Based on an independent study comparing Revolution with a leading product. Please see our website for details. Runtime Revolution, the Runtime Revolution logo and Revolution are trademarks of Runtime Revolution Limited, registered in the United Kingdom. All other trademarks are the property of their respective owners.



WHEN THE SOLUTION MATTERS

Open-aisle e-commerce development

Build Now. Sell Today.

4D Business Kit

All you need to build and host an online store, from shopping to shipping.

- **Secure**
- **Easy to set-up**
- **One integrated solution**
- **E-commerce for every business**
- **WWW.4D.COM/4DBK**

MacTech®

The Journal of Macintosh Technology & Development

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

The MacTech Editorial Staff

Publisher • Neil Ticktin
Managing Editor • Jessica Stubblefield
Online Editor • Jeff Clites

Regular Columnists

Getting Started – Networking
by John C. Welch
Programmer's Challenge
by Bob Boonstra
MacTech Online
by Jeff Clites

Regular Contributors

Vicki Brown, Andrew Stone, Tim Monroe, Erick Tejkowski, Kas Thomas, Will Porter, Paul E. Sevinç, Tom Djajadiningrat, and Jordan Dea-Mattson

MacTech's Board of Advisors

Dave Mark, Jordan Dea-Mattson, Jim Straus, Jon Wiederspan, and Eric Gundrum

MacTech's Contributing Editors

- Michael Brian Bentley
- Derek J. Burks
- Tantek Çelik, Microsoft Corporation
- Marshall Clow
- John. C. Daub
- Tom Djajadiningrat
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Cobalt Networks
- Lorca Hanns, San Francisco State University
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Scott Knaster, Microsoft
- Peter N. Lewis, Stairways Software
- Rich Morin
- John O'Fallon, Maxum Development
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Andrew C. Stone, www.stone.com
- Michael Swan, Neon Software
- Chuck Von Rospach, Plaidworks
- Bill von Hagen
- Eric Zelenka

Xplain Corporation Staff

Chief Executive Officer • Neil Ticktin
President • Andrea J. Sniderman
Controller • Michael Friedman
Production Manager • Jessica Stubblefield
Production/Layout • W2 Graphics
Production/Layout • Darryl Smart
Marketing Manager • Nick DeMello
Events Manager • Susan M. Worley
Network Administrator • Ashutosh A. Gholkar
Accounting • Jan Webber, Marcie Moriarty
Customer Relations • Laura Lane
Susan Pomrantz
Shipping/Receiving • Joel Licardie

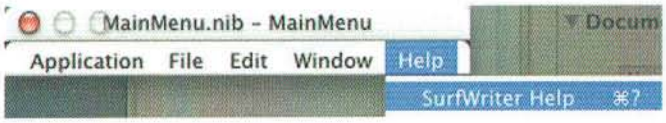
Board of Advisors • Steven Geller, Blake Park, and Alan Carsrud

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

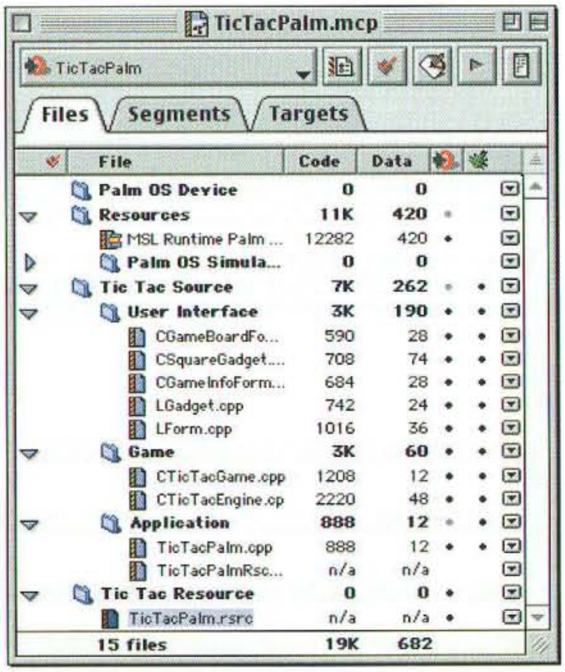
POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

April 2002 • Volume 18, Issue 04



Edit the Help menu item . . . . page 23



File	Code	Data		
Palm OS Device	0	0		
Resources	11K	420	●	
MSL Runtime Palm ...	12282	420	●	
Palm OS Simula...	0	0		
Tic Tac Source	7K	262	●	●
User Interface	3K	190	●	●
CGameBoardFo...	590	28	●	●
CSquareGadget....	708	74	●	●
CGameInfoForm...	684	28	●	●
LGadget.cpp	742	24	●	●
LForm.cpp	1016	36	●	●
Game	3K	60	●	●
CTicTacGame.cpp	1208	12	●	●
CTicTacEngine.cp	2220	48	●	●
Application	888	12	●	●
TicTacPalm.cpp	888	12	●	●
TicTacPalmRsc...	n/a	n/a		
Tic Tac Resource	0	0	●	
TicTacPalm.rsrc	n/a	n/a	●	
15 files	19K	682		

The TicTacPalm Project . . . . page 12

By Ron Davis

Book Review: 3D Game Engine Design

There are a lot of game programming books out there. They use many different methodologies to teach you something about game programming. *3D Game Engine Design* by David H. Eberly is a specialize graphic algorithm book. As the author says in his introduction, most graphic algorithm books either show you code to implement games, or they give you in-depth discussion of 3D algorithms that aren't really useful for the game programmer. This book is in the middle. It is a deep reference of 3D programming, but it focuses on the things you need for the real time graphics of games.

I have to confess much of the math in this book was over my head. It isn't for those of us who didn't pay enough attention to those calculus classes. If this sentence from the introduction of the book scares you, the book isn't for you: "It is assumed that the reader's background includes a basic understanding of vector and matrix algebra, linear algebra, multivariate calculus and data structures."

Even given that caveat I delved on into the book. It is first and foremost a reference. Each aspect of 3D engine development is covered, with no one part being dependent on another. If you do this kind of development, you need this book on your book shelf. There is also a CD that comes with the book that includes a full 3D graphic engine, but, alas, it is not very Mac friendly.

The first chapter is an introduction to the book and how it is organized. The second chapter discusses geometrical methods used to map three dimensional objects onto a two dimensional screen, including transformations, coordinate systems, quaternions, Euler angles and distance methods. There is also a discussion of common shapes like spheres, capsules, cylinders etc.

Chapter 3 discusses the graphic pipeline, which is how a 3D game works, including perspective projection, camera models, culling and clipping. There is also a discussion of surface and vertex attributes including textures, lighting, transparency and fog. According to the summary at the end of the chapter, all of this information is not needed if you are using a 3D API like OpenGL.

No matter what you use for a render, you will have to organize the scene you are going to feed the renderer. The book begins a discussion of these issues with Chapter 4, Hierarchical Scene Representations. It discusses means of organizing objects so they are grouped allowing for easier transformation, collision detection, rendering and persistence.

Picking is the subject of Chapter 5, which is the means of translating a click on the screen to a 3D object in the scene. The author takes a more generalized approach and handles collision detection from any 2D point for a ray into the 3D world. This allows for determining if things like laser beams hit 3D objects.

Chapter 6 covers collision detection. The author notes this can be a very complex problem and that much of the code a game can be taken up with it. He notes at the beginning of the chapter that this needs to be taken into account at design time. The rest of the chapter is dedicated to collisions of various types of objects.

Chapter 7 covers curves which may not at first be obviously used in games, but which the author discusses anyway. This leads to surfaces in Chapter 8, which are a growing part of 3D game engines. The chapter covers different kinds of curves and how to subdivide them.

Chapter 9 is a short chapter on animation covering key frame animation, inverse kinematics and skinning. The author discusses how to create different levels of detail in a 3D scene on demand in chapter 10.

One of the cooler things in 3D to me is terrain generation and that is topic of chapter 11. He discusses an algorithm used to minimize the amount of data and calculations needed to render landscapes.

The last two chapters of the book cover spatial sorting techniques and special effects. The book concludes with an appendix on object oriented infrastructure and numerical methods as wells as a complete glossary.

Ron Davis is a long time Macintosh Software Engineer, having worked for companies like Apple, and Metrowerks on a variety of products from development tools to anti-virus software.

PROFESSIONAL SOFTWARE PROTECTION

HASP®

One Key, Multiple Platforms

www.eAladdin.com

Now your cross-platform software enjoys cross-platform security. By preventing unauthorized software usage and duplication, HASP protects intellectual property rights and increases revenues across the board. A single HASP USB protects Windows, Linux and Mac applications, for reduced development time, simplified shipping logistics and lowered costs. Our cross-platform software protection key benefits your customers too – now they can use your application on multiple platforms with maximum convenience. More developers than ever rely on HASP to increase revenues.

HASP supports:

- Windows 3.x/95/98/ME/2000/NT4/XP
- Linux
- Mac OS 8.6-9.x
- Mac OS X
- TCP/IP, IPX, NetBIOS-based networks

To order your HASP Developer's Kit, visit:
www.eAladdin.com/mactech
or call 800-223-4277 EST
& 800-562-2543 PST, CST, MST

Aladdin®

SECURING THE GLOBAL VILLAGE

By Danny Swarzman

TicTacPalm

Getting started with Palm OS

Preface

Developing a Palm OS application comes naturally to a Mac programmer. You find familiar notions like resources, handle-based memory and an event loop. You can use CodeWarrior for Palm OS as the development environment.

This article presents a simple Palm OS application, TicTacPalm, which illustrates the main features of a Palm OS application. This application simply plays tic tac toe against the user.

The Palm OS Environment

The Palm Device

The Palm Operating System, Palm OS, is used in a growing number of products made by different manufacturers including Palm, Handspring and Sony.

The screen is typically divided into a 160x160 pixel display area, although at least one new device has a slightly longer screen. (Remember when Inside Macintosh admonished developers not to make assumptions about a fixed screen size? That advice holds true for Palm OS as well.) An application can detect stylus activity in this area. The stylus can also be used to enter characters in Palm's custom cursive writing system, *graffiti*, in the area just below the display. All stylus taps enter the system event queue as discrete pen events, similar to the mouse events you are familiar with in Mac OS. The graffiti input comes in the form of graffiti events.

Applications are loaded into the device by the *synchronizing* with a desktop computer through a serial port or IR communication. This is the standard way for a Palm device user to install your program. Some types of programs may be downloaded via a network connection, but we won't consider that approach here.

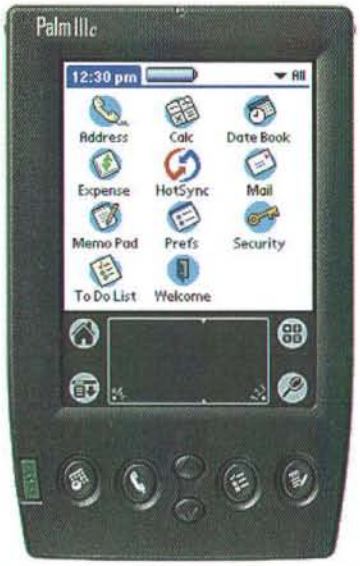


Figure 1. *A typical Palm device. The Application Launcher program provides the equivalent of a "desktop" on a Palm Device.*

Launching and Running Applications

The device shown in **Figure 1** is displaying the equivalent of a desktop. Tapping an icon brings an application to the front. From the user's point of view, all applications are always running. When an application is brought to the front, it looks exactly like it did the last time it was in front.

An application must give up its temporary memory and close down each time another application comes to the front. The closing application can store information about the partially completed document or transaction prior to closing. Later, when the application is activated again it can read the stored information and restore the program state. A special database is used for this purpose, the *preferences*. There are functions in the API designed specifically for reading and writing preferences, greatly simplifying the task.

Danny Swarzman writes programs in JavaScript, Java, C++ and other languages. He also plays Go and grows potatoes. Contact him with comments and job offers. mailto:dannys@stowlake.com, http://www.stowlake.com

Sometimes, one just isn't enough...
DVIator for ADC

... and sometimes it is.
With DVIator and a DVI video card you can add additional ADC displays to a new G4, or upgrade an older Mac with Apple's new flat-panels.

Dr. Bott LLC

www.drbott.com • 877.611.2688 • 503.582.9944

Forms

Like a window on a desktop system, a Palm OS Form object occupies an area of the screen. Due to the small screen sizes on current devices, many forms use the entire visible area. An application may have several forms and switch between them in response to user actions. At any given time, one form is in front and interacting with the user.

In Palm OS parlance, a *window* is any entity in which drawing occurs. Both forms and offscreen bitmaps are windows. Offscreen bitmaps allow you to create a composite image and then transfer it flicker-free to the screen.

Using Constructor, the GUI design tool included with CodeWarrior for Palm OS, you can create forms and populate them with controls. Controls include buttons, checkboxes, lists, etc. Each control is identified by a symbol or name that you specify. Each name will be included in the auto-generated resource file along with a numeric equate. For example, suppose you have a button named New on the form named GameBoard. In the generated header file, Constructor inserts a #define for the symbol GameBoardNewButton. Its value is the numeric ID of the button. Using the symbol (and not the numeric value) in your source code makes the code easier to read and understand.

Controls and Gadgets

Controls include interface elements such as buttons, labels, fields, etc. For most controls, there is a standard behavior provided by the OS. The exception is the *gadget*. The application defines a gadget's behavior, responding when the stylus enters its area and drawing its contents. Most of the time you can use the pre-built controls, and use gadgets only when custom appearance or behavior is required.

Memory

Palm OS uses a handle-based memory management system. These handles can't be treated as pointers to pointers, as is done in Mac OS. Instead, to reference data a program first *locks* a handle. The handle-locking function returns a pointer to the data. You can then use the pointer to transfer data to or from the area. Remember to unlock the handle when you are done using it, or heap fragmentation may result. This becomes even more of an issue on older devices with less memory. If you are aiming for compatibility with older devices and/or older versions of Palm OS, take a close look at the available dynamic heap space versus your program's requirements. The issue exists in a less severe form on modern devices and versions of the operating system.

Events

The operating system maintains a queue of events. Whenever the system requires the services of an application that has been launched, it sends events to the application.

For many events there is a standard response. When the user presses the stylus on the menu bar, the sequence of stylus events is interpreted as a menu command. The menu command then becomes a new event that is placed in the queue.

Some events will appear in your application's event handler, but are really intended for the system. You can ensure that your application hands events to the appropriate Palm OS functions by retaining the AppEventLoop exactly as provided in the Starter.cpp , part of the template code that is generated when you build a new project.

Everything relating to your application from launch to "shutdown" will arrive in the form of events. For example, at launch time an application-defined function, called from the main event loop, processes a request to load a form. That function assigns a callback function to the form to assist in the handling of events occurring in controls in that form. Once the form is loaded the callback function handles subsequent events.

A gadget, the custom user interface object discussed earlier, can have a callback function too. The application assigns a callback to each gadget when the form is opened. This is how a gadget responds to system requests to draw itself, and to handle taps in the gadget.

DEVELOPING FOR PALM OS

CodeWarrior and Alternatives

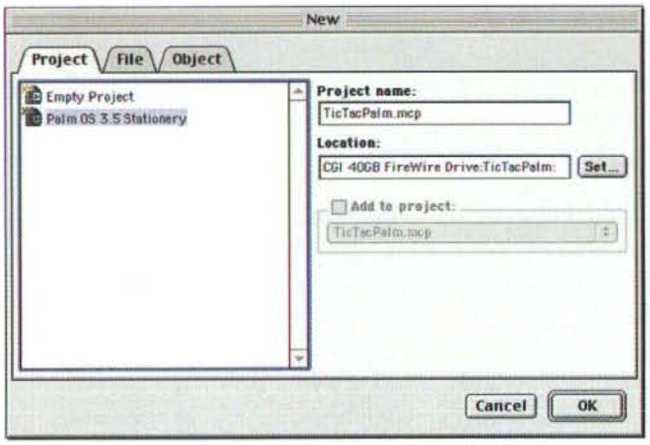
The vast majority of applications are developed using CodeWarrior However, the official Palm SDK, available on the Palm web site, works with many development environments. Other tools that are available to create Palm applications include a gnu compiler and a translator from Java bytecode to assembler for the handheld.

The sample code in this article was developed using CodeWarrior for Palm OS, Version 7. It comes with everything you need to develop for Palm OS 3.5, including the Palm SDK and related documentation. Current devices run Palm OS 3.5 and Palm OS 4.0, although with proper attention to detail you can build an application supporting earlier versions of the operating system.

The CodeWarrior package includes two disks – one to develop on the Mac, one to develop on Windows. The sample code accompanying this article was developed on the Mac.

Creating a Project

Use the New command from the File menu. In the dialog that appears, enter a name for the new project and choose Palm OS 3.5 Stationary.



Empty Project
Palm OS 3.5 Stationery

Project name: TicTacPalm.mcp

Location: CGI 40GB FireWire Drive:TicTacPalm:

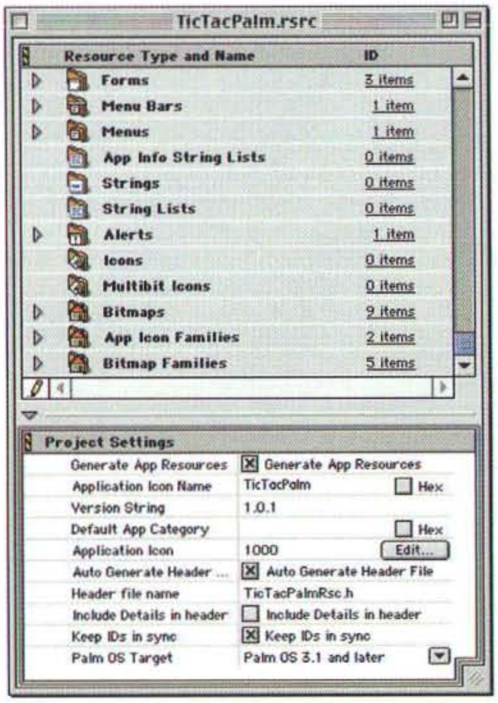
Add to project: TicTacPalm.mcp

Figure 2. Creating the project in CodeWarrior.

A second dialog will appear, giving you a few choices for the template project. For the sample code, Palm OS C++ App was used. CodeWarrior then generates a 'Starter' project. This project produces a do-nothing application to use as a framework on which to build. You need to rename some things if your project's name isn't to be 'Starter'. I just replaced every 'Starter' occurrence with 'TicTacPalm' in the code, file names, etc.

Constructor

Constructor for Palm is an interactive resource editor included with CodeWarrior for Palm. Constructor creates a collection of resources that contribute to the appearance of the application, including menus, forms and icons. In addition to editing a resource file, Constructor generates a header file that defines symbols for all the resources in the resource file.



Resource Type and Name	ID
Forms	3 items
Menu Bars	1 item
Menus	1 item
App Info String Lists	0 items
Strings	0 items
String Lists	0 items
Alerts	1 item
Icons	0 items
Multibit Icons	0 items
Bitmaps	9 items
App Icon Families	2 items
Bitmap Families	5 items

Project Settings	
Generate App Resources	[✓] Generate App Resources
Application Icon Name	TicTacPalm [ ] Hex
Version String	1.0.1
Default App Category	[ ] Hex
Application Icon	1000
Auto Generate Header ...	[✓] Auto Generate Header File
Header file name	TicTacPalmRsc.h
Include Details in header	[ ] Include Details in header
Keep IDs in sync	[✓] Keep IDs in sync
Palm OS Target	Palm OS 3.1 and later

Figure 3. The Constructor project window.

Tri-platform Calendaring

OS X Carbon
Mac OS 7.1 – 9.x
Windows 95 – 2000

New!

CalendarMonster 1.2

Multi-platform calendar

- **End-user Calendar**
  The best FMP calendar experience
  - Fast, fast, fast!!!
  - Platform-appropriate user experience
  - Highly scalable
- **Excellent Developer Support**
  Quickly and easily integrate with your FileMaker Pro projects
  - 100% live data, 100% modular, no plug-ins
  - FileMaker Pro 4 or 5 (including 5.5)
  - As little as 2 ScriptMaker calls
  - Dedicated developers' utility

only $49! per user

Developer and volume licensing available

ASCENDING TECHNOLOGIES

monster@asctech.com

Download your own FREE fully-functional demo

http://www.asctech.com

Debugging, Simulator and POSE

A Palm application can run in one of these environments:

- the Palm Operating System Emulator (POSE)
- the Simulator
- the device itself

The sample project, like the Starter project has two targets: TicTacPalm and TicTacPalmSim.

TicTacPalm generates a Palm OS application that can be run in POSE or on an actual device. CodeWarrior provides a Palm OS debugger to step through the object code for this target.

For the TicTacSim target, a collection of simulator libraries is included in the project. When the project is built, a MacOS application is created. CodeWarrior's MacOS debugger can be used to step through this application. This is the easiest way to debug. There are limitations to this approach in that you don't have access to other applications or permanent data, which is not significant for our game project, but may be an issue for more complex applications.

There are a couple of strange rituals that you need to follow when using the simulator. The first time you run a program with the simulator, a dialog asks you to find the resources file. You must locate :CW for Palm OS Platform 7.0:Metrowerks CodeWarrrior:Palm OS Support:Simulator:SystemResources. You may notice that the simulator window sticks to the menu bar on your screen. To fix that, remove the Palm OS Simulator Preference file from the System Folder's Preferences subfolder. You only need to do these things once or twice when you are first using the product.

Most of the time, POSE is the preferred debugging platform. POSE is a desktop application that emulates the entire Palm OS device. It allows you to switch between applications, and even retain information between runs. You set up POSE to emulate a particular Palm OS device by copying the ROM from your device or by getting the ROM file for another device from the Palm web site. The POSE application must be running when you run the TicTacPalm target. Use the CodeWarrior debugger for Palm OS with this target.

Debugging while running on the device itself is possible but less convenient. It wasn't needed for the sample application. When the code works in simulation, load the .prc file onto the device as you would any Palm application.

USER'S VIEW OF TICTACPALM

Forms

TicTacPalm includes these forms:

- Game Board
- Game Info
- About

Most of the action occurs on the game board (see **Figure 4**). The user taps on a square to make an 'X' move. The program responds by playing an 'O'. There are two buttons: **New** causes the Game Info dialog to be displayed, while **Clear** erases all the plays.

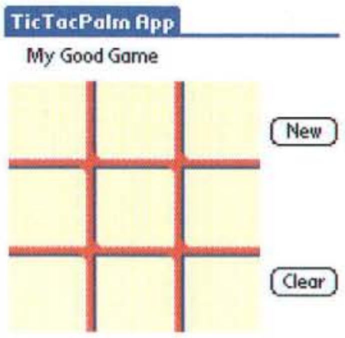
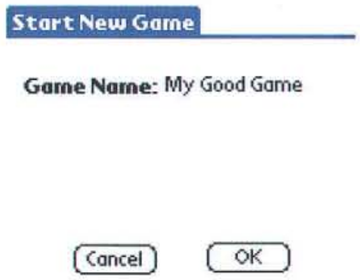


Figure 4. *The game board.*

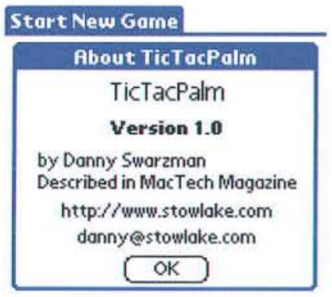
The game information form has one field for the user to enter a name for the game. There is an **OK** button and a **Cancel** button. OK starts a new game with the name in the name field. Cancel returns to the game board without altering the name or the position.


Start New Game

Game Name: My Good Game

Figure 5. *Game info form.*

The About form is a modal dialog that appears in response to the About menu command.


Start New Game

About TicTacPalm

TicTacPalm

Version 1.0

by Danny Swarzman
Described in MacTech Magazine
http://www.stowlake.com
danny@stowlake.com

Figure 6. *The About form.*

A New System Has Arrived.

No Mac is complete without Timbuktu Pro, the premier remote control and file transfer software for Mac OS for over ten years.

No network is complete without the smart systems management of netOctopus.

Both tools are newly rewritten for OS X, bringing the power and simplicity of Netopia software to the world's most advanced operating system. For deployment, training and support of all your Macs (and PC's!) Timbuktu and netOctopus are indispensable.

Are you ready to migrate? We're ready to take you there.

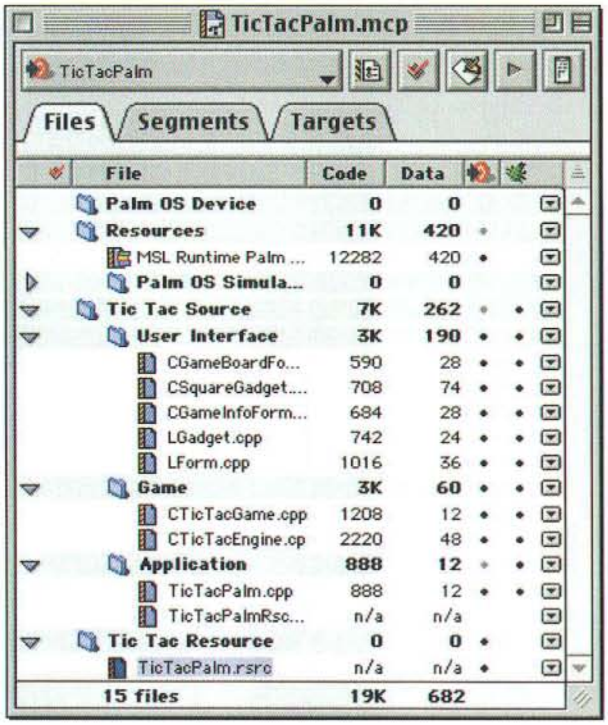
macosxready.com

Timbuktu Pro   netopia.   netOctopus

Source Files

The source files are organized into three subgroups:

- Application
- User Interface
- Game



TicTacPalm.mcp

TicTacPalm

Files | Segments | Targets

	File	Code	Data		
	Palm OS Device	0	0		
▽	Resources	11K	420	●	
	MSL Runtime Palm ...	12282	420	●	
▷	Palm OS Simula...	0	0		
▽	Tic Tac Source	7K	262	●	●
▽	User Interface	3K	190	●	●
	CGameBoardFo...	590	28	●	●
	CSquareGadget....	708	74	●	●
	CGameInfoForm...	684	28	●	●
	LGadget.cpp	742	24	●	●
	LForm.cpp	1016	36	●	●
▽	Game	3K	60	●	●
	CTicTacGame.cpp	1208	12	●	●
	CTicTacEngine.cp	2220	48	●	●
▽	Application	888	12	●	●
	TicTacPalm.cpp	888	12	●	●
	TicTacPalmRsc...	n/a	n/a		
▽	Tic Tac Resource	0	0	●	
	TicTacPalm.rsrc	n/a	n/a	●	
	15 files	19K	682		

***Figure 7.** The TicTacPalm Project.*

Application

When you generate a project using the project stationary, one of the files, Starter.cpp, contains the main program for the application. In the sample application, Starter.cpp is replaced by TicTacPalm.cpp. It has seven functions:

- AppHandleEvent
- AppStart
- AppStop
- RomVersionCompatible
- AppEventLoop
- TicTacPalmMain
- PilotMain

The contents of the first three are new in TicTacPalm.cpp. Of the other four, only TicTacPalmMain has been changed from the original StarterMain in Starter.cpp. One line of StarterMain has been commented out.

```
FrmGotoForm ( MainForm );
```

In the sample application, the first form is loaded in AppStart rather than the main program.

Three variables of file scope are defined in TicTacPalm.cpp. Their referenced objects are created in AppStart and deleted in AppStop. They are:

```
static CGameBoardForm *fGameBoardForm = NULL;
static CGameInfoForm *fGameInfoForm = NULL;
static CTicTacGame *fGame = NULL;
```

The first two handle the two different forms. The game object contains all the data about the current game that is in dynamic RAM. Each of the form objects is initialized with a pointer to the game.

AppStart

```
// static
Err AppStart(void)
{
  UInt16 prefsSize;
  TicTacPalmPreferenceType preferences;

  // Read the saved preferences / saved-state information.
  fGame = new CTicTacGame ();
  fGameBoardForm = new CGameBoardForm( fGame );
  fGameInfoForm = new CGameInfoForm ( fGame );

  prefsSize = sizeof(TicTacPalmPreferenceType);
  if (PrefGetAppPreferences(appFileCreator, appPrefID,
    &preferences, &prefsSize, true) != noPreferenceFound)
    {
    fGame->SetPrefs ( &preferences );
    }
  // Display the first form.
  FrmGotoForm ( fGame->GetForm() );
  return errNone;
}
```

Listing 1. AppStart

The preferences record stores information used to reconstruct the last state of the program so that it will appear to the user that the program wasn't interrupted at all. If there is no preferences record to read, a default preferences record is constructed.

AppStart creates fGame and initializes it with the values in the preferences. It then brings forward a form, chosen according to the last state of the application as recorded in the preferences.

AppHandleEvent

```
// static
Boolean AppHandleEvent(EventPtr eventP)
{
  return eventP->eType == frmLoadEvent &&
    LForm :: Load ( eventP->data.frmLoad.formID );
}
```

Listing 2. AppHandleEvent

AppHandleEvent sends the load event to the LForm class (which is discussed later).

AppStop

```
// static
void AppStop(void)
{
  FrmCloseAllForms ();
  TicTacPalmPreferenceType preferences;
  fGame->GetPrefs ( &preferences );
  PrefSetAppPreferences (appFileCreator, appPrefID,
    appPrefVersionNum, &preferences,
```

Your Data Isn't an Important Part of the Job — It *IS* the Job.

VXA® FireWire

The High-Performance Tape Storage Solution For Apple Users

"I can stick my entire hard drive onto a tape 7 times —that's just plain cool!"

Other backup media, such as CD-RWs and DVD-RAMs, simply can't compare to VXA-1 tape technology when it comes to capacity, transfer rate and overall price. Fast, economical and easy-to-use, VXA FireWire offers:

Ultimate File Security
- 100% Data Restore and Interchange
- Plug-and-Play Connectivity

Sharable Media
- Multi-Gigabyte File Transport
- Portable, Hot-Pluggable

Real Time Digital Video

Cross-Platform Compatibility
- FireWire/IEEE1394/iLink Supported By All Major Manufacturers

LOWEST COST per MB Of ANY Technology!

Technology	Capacity in MBs	Transfer Rate in MB/sec	Price per MB
VXA Tape	33000	3	$0.03
Imation Travan	10000	1	$0.05
DVD RAM	9400	2.7	$0.05
CD RW	700	1.9	$0.50
Zip	250	1.2	$0.76

Call Exabyte sales at 1-800-774-7172 or visit us on the web at www.exabyte.com

Tape Storage By

VXA® Exabyte®

```
    sizeof (preferences), true);

  if ( fGameBoardForm )
    delete fGameBoardForm;
  if ( fGameInfoForm )
    delete fGameInfoForm;
  if ( fGame )
    delete fGame;
}
```

Listing 3. AppStop

When the application receives a stop event, AppStop saves into the preference record the state of things as recorded in gGame, and deletes the objects that were created in AppStart.

Supporting Forms

Associated with an active form is a callback function that processes events sent to the form. The classes CGameBoardForm and CGameInfoForm handle events for the two types of forms. These classes are subclasses of LForm.

LForm :: DispatchEvent

```
// static
Boolean LForm :: DispatchEvent ( EventType *inEvent )
{
  // Identify the event type and respond to it accordingly.

  Boolean handled = false;
  if ( sActiveLForm )
    switch ( inEvent->eType )
    {
      case frmOpenEvent :
        // Do follow up after open event
        FrmDrawForm ( FrmGetActiveForm() );
        handled = sActiveLForm->Open();
        break;
      case frmCloseEvent :
        // Close stuff before closing if necessary
        handled = sActiveLForm->Close();
        break;
      case menuEvent :
        handled = sActiveLForm->
          DoCommand ( inEvent->data.menu.itemID, NULL );
        break;
      case ctlSelectEvent:
        handled = sActiveLForm->
          DoSelect ( inEvent->data.ctlSelect.controlID,
            NULL );
        break;
      default :
        handled = false;
        break;
    }
  return handled;
}
```

Listing 4. LForm::Dispatch Event

LForm :: Load

```
// static
Boolean LForm :: Load ( UInt16 inFormID )
//
// The Palm OS has requested that a form be loaded.
// If the form can be loaded, mark the appropriate
// LForm
{
  Boolean handled = false;
  FormPtr formPointer = FrmInitForm ( inFormID );
  if ( formPointer )
  {
    sActiveLForm = IDToLForm ( inFormID );
    if ( sActiveLForm )
    {
      FrmSetActiveForm ( formPointer );
      FrmSetEventHandler ( formPointer, &DispatchEvent );
      handled = true;
    }
  }
  return handled;
}
```

Listing 5. LForm::Load

When the system function FrmGotoForm is called to switch forms, the application sends the function to the class function, LForm::Load. That function finds the LForm object that will handle the form according to the ID requested. It establishes the event handler for that class as the handler for form events.

When events arrive for a form, the class function LForm::DispatchEvent, sends the event to one of the virtual functions, DoCommand, DoSelect, Open or Close. Subclasses need only override these functions.

Game Board Form

This form displays the name of the current game, the playing surface and two buttons. One button clears the board. The other switches to the game information form.

CGameForm::Open

```
// virtual
Boolean CGameBoardForm :: Open ()
{
  // Set up all the gadgets as CSquareGadget objects
  FormPtr form = FrmGetActiveForm();
  UInt16 numberOfObjects = FrmGetNumberOfObjects ( form );
  for ( UInt16 j = 0; j < numberOfObjects; j++ )
  {
    FormObjectType *objectPointer =
      (FormObjectType*)FrmGetObjectPtr( form, j );
  FormObjectKind objectKind = FrmGetObjectType ( form, j );
    if ( objectKind == frmGadgetObj )
    {
      FormGadgetType *gadget =
(FormGadgetType*)objectPointer;
  CSquareGadget *squareGadget = new CSquareGadget ( mID, j,
        gadget, mGame );
    }
  }
  GameNameType gameName;
  mGame->GetName ( gameName );
  SetFieldText ( GameBoardGameNameField, gameName );
  FrmDrawForm ( form );

  // Also mark the game that this form is active.
  mGame->SetForm ( GameBoardForm );
  return true;
}
```

Listing 6. CGameBoardForm::Open

Open creates a collection of objects to handle the gadgets that will be used in the form. These objects are not deleted in CGameBoardForm. They delete themselves as we'll see in the section on gadgets.

The list of controls in the form is scanned to find controls that are gadgets. As each gadget is found, a LgameSquareGadget object is created to handle its events.

CGameBoardForm::DoSelect

```
// virtual
Boolean CGameBoardForm :: DoSelect ( UInt16 inCommand, void
```

AS A PROGRAMMER OR SYSADMIN, YOU'VE GOT BETTER THINGS TO DO THAN FIGHT WITH A WEB SITE

Mgi

Searchable Databases
Online Stores
Info Baskets
Guestbooks
Banner Ads
Credit Card Transactions
Counters
Password Protection
Surveys
Quizzes
Form Processing
Conditionals
Math
Cookies
Date and Time
File Includes
File Writes
Web Based Email
Classified Ads
Discussion Groups
Web Chat
Affiliate Tracking
I/O Transactions
Δ Time Calculations
PGP Encryption
Token Tracking
Calendars
Random Rotation
Send Mail
E - Cards
Credit Bank

It's time for you to take a look at MGI

MGI, a plug-in to 4D's award-winning server, WebSTAR, is designed to provide functionality to otherwise static web sites, whether for a private intranet or enterprise - class ISP/ASP. MGI was specifically developed to be used by web graphic designers with no scripting or programming experience. If you know HTML, you know MGI. And if you are a programmer, you can learn MGI by the time your pizza is delivered. You can try out MGI for free – right now – without obligation by downloading a fully - functioning demo at :

http://www.pageplanetsoftware.com

page planet software inc

SOFTWARE BUILT WITH YOU IN MIND

PagePlanet Software Inc., 3252 Octavia Street, Raleigh, NC 27606 919.233.1770

```
*/*inParam*/ )
{
  Boolean handled = false;
  switch ( inCommand )
  {
    case GameBoardClearButton:
      mGame->Clear();
      FormPtr activeForm = FrmGetActiveForm();
      FrmDrawForm ( activeForm );
      handled = true;
      break;
    case GameBoardNewButton:
      FrmGotoForm(GameInfoForm);
      handled = true;
      break;
  }
  return handled;
}
```

Listing 7. CGameForm::Open

CGameBoardForm::DoCommand

DoSelect handles the two buttons: Clear and New. For the Clear button, it just tells the game object to clear its data and then redraws the form. FrmDrawForm will call the gadget handlers to do the drawing. The gadget handlers will then retrieve any needed data from the game object as part of the redraw operation.

GameInfoForm

With this form the user enters a name for the game. When the user taps on the OK button, the current game is cleared and a new game is started and control is switched to the game board form. If the user taps Cancel, control also returns to the game board form, but the name is not changed and the game resumes.

The user could also switch to another application. In that case, the name field is restored as it was, with a name that the user has changed, but not confirmed by tapping OK.

CGameInfoForm :: Open

```
// virtual
Boolean CGameInfoForm :: Open()
{
  // Inform the game that the info form is active.
  mGame->SetForm ( GameInfoForm );

  GameNameType name;
  // If the last open form was a game info form then use the name
  // that appeared on that form.
  mGame->GetUnconfirmedName ( name );

  // Otherwise use the name of current game.
  if ( StrLen ( name ) == 0 )
    mGame->GetName ( name );
  mGame->SetUnconfirmedName ( name );
  SetFieldText ( GameInfoNameFieldField, name );
  FieldType *field = GetField ( GameInfoNameFieldField );
  FldDrawField ( field );

  mOKWasPressed = false;
  mCancelWasPressed = false;
  return true;
}
```

Listing 8. CGameInfoForm::Open

If the form was replaced because the user changed applications, then an unconfirmed name is stored in the game object and the unconfirmed name will be loaded into the field. Otherwise, the name of the current game will be loaded into the field.

CGameInofForm::Close

```
// virtual
Boolean CGameInfoForm :: Close()
{
  GameNameType name;
  if ( mCancelWasPressed )
    // Don't care about what was in the field.
    mGame->SetUnconfirmedName ( "" );
  else
  {
    if ( GetFieldText ( GameInfoNameFieldField, name ) )
      if ( mOKWasPressed )
      {
        mGame->Clear();
        mGame->SetName ( name );
        mGame->SetUnconfirmedName ( "" );
      }
      else // switched applications
        mGame->SetUnconfirmedName ( name );
  }
  // Return false to tell the OS to clean up the form
  // in the usual way after we have extracted the info.
  return false;
}
```

Listing 9 CGameInfoForm::Close

Close sets the name and unconfirmed name fields in the game according to two flags: mOKWasPressed and mCancelWasPressed.

CGameInfoForm::Open

```
// virtual
Boolean CGameInfoForm :: DoSelect ( UInt16 inCommand, void
*/*inParam*/ )
{
  Boolean handled = false;
  switch ( inCommand )
  {
    case GameInfoOKButton:
      mOKWasPressed = true;
      FrmGotoForm(GameBoardForm);
      handled = true;
      break;
    case GameInfoCancelButton:
      mCancelWasPressed = true;
      FrmGotoForm(GameBoardForm);
      handled = true;
      break;
  }
  return handled;
}
```

Listing 10 CGameInfoForm::DoSelect

DoSelect sets the appropriate flags according to which button was pressed. It then switches to the game board form.

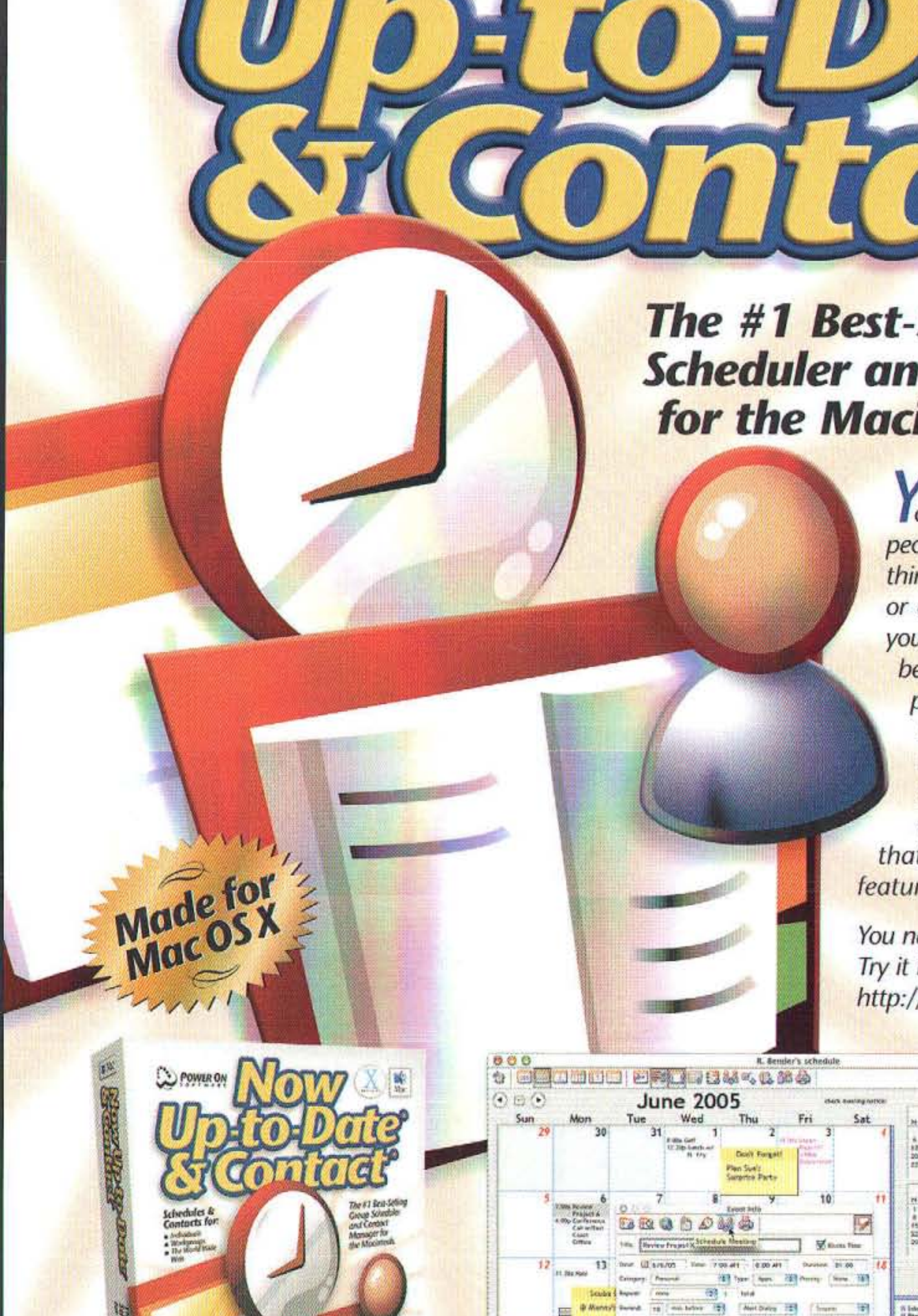
A Gadget for a Square

Instead of handling *events* as is done for forms, a gadget callback handles *commands*. One type of command is an event command. So you need two levels of dispatch: one at a higher-level for events, and one at a lower-level for commands.

In TicTacPalm, there is a gadget for each of the nine squares in the game board. The class LGadget handles the dispatch. Its subclass, CSquareGadget contains the code to respond to actions.

The Palm data type for gadgets contains a field for application defined data. Unfortunately, the argument passed to the handler is a pointer to an *opaque* structure. Palm documentation warns you not to directly access its fields. But they don't provide the accessor functions needed to conveniently

Now Up-to-Date® & Contact®



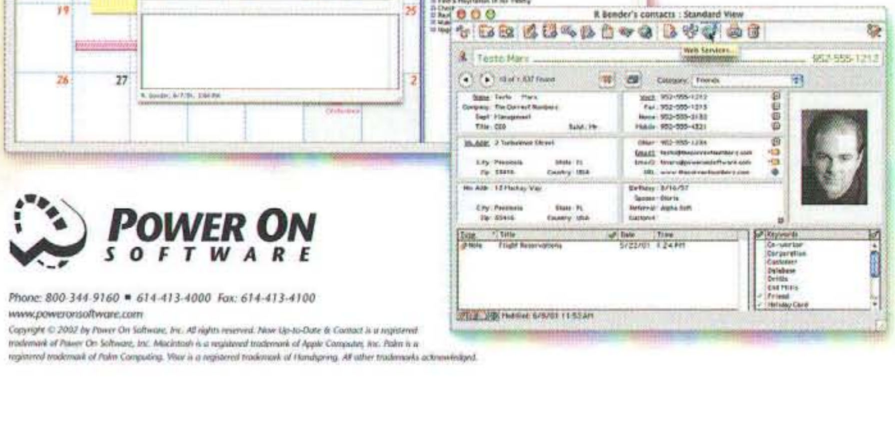
The #1 Best-Selling Group Scheduler and Contact Manger for the Macintosh!

You're a busy person. You've got things to do, people to see, and a seemingly infinite list of things to keep track of. Whether it's just yourself or a department at a Fortune® 500 company, you (and everyone that works with you) need to be organized. Maybe you've tried other ways: paper planners, lots of sticky notes, other software. But now you can't afford to waste any more time or effort. You need a serious solution. One that's so easy to use that you can get started right out of the box. One that will grow with you. One that has the features and capabilities you need.

You need Now Up-to-Date & Contact! Try it FREE! Get your free 30-day trial at http://www.poweronsoftware.com

Power On Software
Now Up-to-Date & Contact
Schedules & Contacts for:
The #1 Best-Selling Group Scheduler and Contact Manager for the Macintosh
NEW VERSION!

Supports Mac OS 8.6-OS X



Fully Mac OS X & Palm Compatible.

Available @ Office DEPOT

POWER ON SOFTWARE

Phone: 800-344-9160 ■ 614-413-4000 Fax: 614-413-4100
www.poweronsoftware.com

get the application data, making the data field unusable. In Palm OS 4.0 an accessor function was added but it doesn't work with all the commands that may be sent to the handler.

The workaround is to search all the gadgets looking for a matching pointer.

LGadget::LGadget
```
LGadget :: LGadget ( UInt16 inFormID, UInt16 inGadgetIndex,
  FormGadgetType *inGadget )
: mFormID ( inFormID ), mGadgetIndex ( inGadgetIndex ),
  mGadget ( inGadget )
{
  FormType *formPointer = FrmGetFormPtr ( mFormID );
  FrmSetGadgetHandler ( formPointer,
mGadgetIndex, &DispatchCommand );
  mPrevious = sLast;
  sLast = this;
}
```

Listing 11. LGadget::LGadget

The constructor stores a pointer to the data structure for the form and sets a class function of LGadget to be the handler.

LGadget.cpp
```
// static
Boolean LGadget :: DispatchCommand ( FormGadgetType
*inGadget,
  UInt16 inCommand, void *inParam )
{
  Boolean success = false;
  LGadget *lGadget = FindLGadget ( inGadget );
  success = lGadget && lGadget->DoCommand ( inCommand,
inParam );
  return success;
}
```

Listing 12. LGadget::DispatchCommand

DispatchCommand is the class function that receives the command. It finds the LForm object to handle that command.

LGadget::DoCommand
```
Boolean LGadget :: DoCommand ( UInt16 inCommand, void
*inParam )
{
  Boolean handled = false;
  switch ( inCommand )
  {
    case formGadgetDrawCmd :
      Draw();
      mGadget->attr.visible = true;
      handled = true;
      break;
    case formGadgetHandleEventCmd :
      handled = HandleEvent ( (EventType*)inParam );
      break;
    case formGadgetDeleteCmd :
      delete this;
      handled = true;
      break;
    case formGadgetEraseCmd :
      handled = false;
      break;
    default :
      handled = false;
      break;
  }
  return handled;
}
```

Listing 13. LGadget::DoCommand

DoCommand determines the type of command and forwards it along. The delete command gets sent immediately before the enclosing form gets deleted. This handler deletes the LForm object in response.

LGadget::HandleEvent
```
Boolean LGadget :: HandleEvent ( EventType *inEvent )
{
  Boolean handled = false;
  switch ( inEvent->eType )
  {
    case frmGadgetEnterEvent :
      // A pen down event has been passed to the form.
      handled = true;
      Tapped();
      break;
    case frmGadgetMiscEvent :
      // The application, not the system generates
      // this event. Not used in the sample code,
      // include to allow for future modifications.
      handled = false;
      break;
    default :
      // Should never get here.
      handled = false;
      break;
  }
  return handled;
}
```

Listing 13. LGadget::HandleEvent

CSquareget

For each of the squares in the game there is a gadget. For each gadget there is an LSquareGadget object. Squares are numbered 0 through 8. When the user taps a game square, Tapped checks if the square is empty and tries to play an X at that location. It displays the current object and then tells the LSquareGadget object corresponding to the O response to draw itself.

Images of the X and O characters, and an empty field image, are stored as bitmaps. Draw copies the appropriate bitmap to the active form.

CSquareGadget::CSquareGadget
```
CSquareGadget :: CSquareGadget( UInt16 inFormID,
  UInt16 inGadgetIndex, FormGadgetType *inGadget, CTicTacGame
*inGame )
: LGadget ( inFormID, inGadgetIndex, inGadget )
{
  FormPtr form = FrmGetFormPtr ( inFormID );
  UInt16 objectID = FrmGetObjectId ( form, inGadgetIndex );
  mGame = inGame;
  mSquare = SquareNumber ( objectID );
  sSquareGadget[mSquare] = this;
}
```

Listing 14. CSquareGadget::CSquareGadget

The constructor uses the object ID of the gadget to determine the number of the square. That is the number that will be used in the game logic to interpret the play.

CSquareGadget::Draw
```
// virtual
void CSquareGadget :: Draw ()
{
  Coord x;
  Coord y;
  FormType *formPointer = FrmGetFormPtr ( mFormID );
  FrmGetObjectPosition
     ( formPointer, mGadgetIndex, &x, &y );
  UInt16 bitmapID = EmptySquareBitmap;
  PlayerT contents = mGame->Get ( mSquare );
  switch ( contents )
  {
     case kX :
        bitmapID = XBitmap;
        break;
     case kO :
        bitmapID = OBitmap;
        break;
     case kPlayerEmpty :
     default:
        bitmapID = EmptySquareBitmap;
        break;
  }
  MemHandle resource = DmGetResource
     ( bitmapRsc, bitmapID );
  BitmapPtr bitmapResource =
     (BitmapPtr)MemHandleLock ( resource );
  Err error = DmGetLastErr();
  if ( !error && bitmapResource )
  {
     WinDrawBitmap ( bitmapResource, x, y );
     error = MemHandleUnlock ( resource );
     DmReleaseResource ( resource );
  }
}
```

Listing 15. CSquareGadget::Draw

The content of the square is determined from the game object. According to the content, select a resource with the X, O or blank bitmap.

CSquareGadget::Tapped

```
// virtual
void CSquareGadget :: Tapped ()
{
  UInt16 oResponse = kNoPosition;
  if ( mGame && mGame->Get ( mSquare ) == kPlayerEmpty )
  {
     oResponse = mGame->PlayX ( mSquare );
     Draw();
  }
  if ( oResponse != kNoPosition )
  {
     CSquareGadget *cSquareGadget =
        sSquareGadget[oResponse];
     cSquareGadget->Draw();
  }
}
```

Listing 16. CSquareGadget::Tapped

When the user taps on a game square, this function responds by displaying an X. It gets the next O move from the game object and calls the draw function for the CSquareGadget for that square.

Conclusion

Developing for the Palm is easy to do with CodeWarrior and POSE. The Palm OS is continually evolving and making many housekeeping chores easier, such as improved bitmap support. Right now, the API reminds me of the early days of Macintosh. The application programmer must be concerned with a lot of messy details. I find that I miss having an application framework and feel the need to head in that direction.

The need to be able to fold up the tent whenever the user switches applications adds an interesting twist to the job of programming for the Palm OS. Here, too, an application framework would come in handy.

TicTacPalm is a fairly simple application. Additional features might include the ability to store and recall games, beam games between devices, and back games up to a desktop machine. Feel free to use the code presented here as the basis for writing an improved version of TicTacPalm, or possibly some other game.

References

The Palm web site contains tons of information and links to related sites.

http://www.Palm OS.com/dev/

To register a creator code just click on the list item Creator ID. As on Mac OS, creator IDs uniquely identify applications.

The documentation in the Palm SDK is fairly good. There is a Reference manual and a Programmers' Guide. Some third-party books on Palm programming are available. *Palm Programming* by Neil Rhodes and Julie McKeehan is a good introduction, though the material in it is quite dated.

Metrowerks has a demo version of CodeWarrior for Palm that you can use to get started, available at:

http://www.metrowerks.com/products/palm/demo

There is a free online course in Palm programming offered by Metrowerks. It's at:

http://www.codewarrioru.com/CodeWarriorU

Credits

Thanks to Victoria Leonard for graphic resources. Thanks to Bob Ackerman and Victoria Leonard for reviewing the text.

Webmasters! Start your web hosting business with edition.net reseller hosting!

- dedicated platforms
- single and multi-domain hosting
- your choice of Mac OS or Linux platforms
- FileMaker Pro hosting featuring Lasso
- QuickTime streaming server
- Our web-based control panels make it easy to
- deploy your multi-domain server

edition.net – the Fine Arts Network

http://www.edition.net • info@edition.net
+1 (877) 225-3821 toll free • +1 (562) 430-5953 international

By Gordon R. Meyer

Apple Help & AppleScript Studio

Providing Help Systems for AppleScript Studio-based Applications

Apple Help and Aqua

AppleScript Studio is an exciting prospect for Scripters – we are no longer limited to creating the usual droplets or faceless utilities commonly associated with scripting language development.

But power and opportunity come at a price. AppleScript Studio development also "raises the bar" on expectations for how your application looks and behaves. Like any other application on Mac OS X, following the Aqua Human Interface Guidelines will ensure your new application provides an excellent user experience. This article focuses on one aspect of Aqua – providing onscreen user assistance with Apple Help.

Virtually every application needs a help system, and using Apple Help is an easy and fundamental step in adopting Aqua. To get started all you need is a set of HTML help files and this article. Aside from the time spent writing the help content, you can implement a working help system in minutes.

What is Apple Help?

Apple provides the Help Viewer application for displaying your help content. The Help Viewer is optimized for providing onscreen help. It displays HTML 3.2, any QuickTime media you care to use, and it can run AppleScript automations to assist users in accomplishing complex or common tasks. (Because AppleScript Studio-based applications are already scriptable by default, there is a lot of opportunity to enhance your help with useful automations. Something to consider after you've gotten your first-cut at Help implemented.)

The Help Viewer also provides a built-in search engine that quickly gives relevancy ranked search results for your help, and all the help installed on the user's computer. Shortly, you'll learn how to ensure your help content is searchable.

When your application adopts Apple Help, it will be automatically listed in the Help Center. The Help Center allows users to view and search an application's help without having to open the application. It's a great way to find out which applications can handle a special task that you have in mind.

There are more Apple Help features such as automatically building "table of contents" files or the ability to mix locally-stored and Internet-based content. For information about them, see *Providing User Assistance with Apple Help*. If you prefer to watch and listen instead of reading, the WWDC 2001 streaming video of the Apple Help session also introduces all the features and capabilities.

What type of information should you include in your help? Making good onscreen help is a whole article by itself, but the "Help" chapter in *Aqua Human Interface Guidelines* has some discussion, as does the WWDC 2001 session. If you would like some guidance consider using these resources.

For the purposes of this article, don't worry about all the bells and whistles, just get your content into HTML 3.2 format. You can repurpose existing files from your Web site, or use whatever HTML authoring tool you normally use and create the help from scratch.

Preparing Your Help Files

After you've written your help, you need to make one minor addition to identify the "start page" (or "home page) for your help book and provide the title of your help system for the Help Center.

You do both by adding one META tag to the start page for your book. This is typically the table of contents page, or the splash page, or whatever page you want users to see when the Help Viewer opens. If you're re-using content from your Web site, it might be the index.html file, for example.

Listing 1: AppleTitle example

```
<html>
<head>
<meta name="AppleTitle" content="MyAppName Help">
  [...]
</head>
<body>
  [...]
</body>
</html>
```

Gordon Meyer (gordon@apple.com) works on Apple Help and related instructional products.

Dear Burglar,
The key is under the mat. Beer and sandwiches are in the fridge.
♡ Greg & Patty

We're Easier.

Create anything from prototypes to full professional applications. Just drag and drop interface elements while REALbasic handles the details. You concentrate on what makes your stuff great — your ideas! REALbasic compiles native applications for Macintosh, Mac OS X and Windows without platform-specific adjustments. It's the powerful, easy-to-use tool for creating your own software. Each version of your software looks and works just as it should in each environment.

Complex problems shouldn't require complex solutions. The answer is REALbasic.

REALbasic

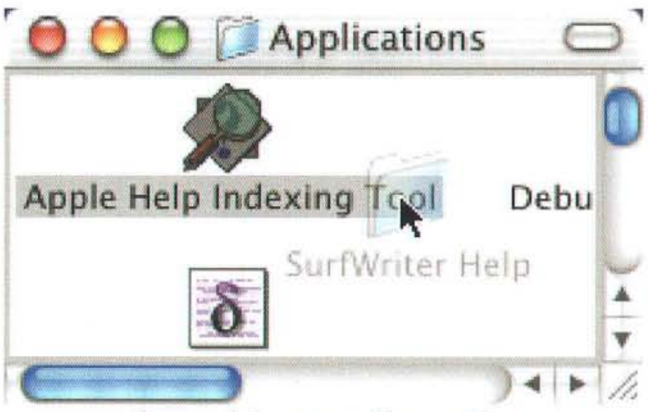
Download a free demo. **www.realbasic.com**

The content of the AppleTitle tag is the name of your help book. This is the name that gets listed in the Help Center, and it's how Apple Help identifies which book to open from your application's help menu. The Aqua HI Guidelines recommend that you use the form "*MyAppName Help*".

Making your help searchable

The search engine in Help Viewer is fast. It can search hundreds of pages of help in just seconds, returning a list of the most relevant topics in your help book. To accomplish this it uses a pre-generated index file. To make your help searchable, which is strongly recommended for the best user experience, you use the Apple Help Indexing Tool. Please don't forget this important step, users get frustrated if they can't locate information by searching. (Indexing is also required if you're using some of the advanced features, such as Anchor Lookup, as described in the Apple Help documentation.)

The Apple Help Indexing Tool is installed in /Developer/Applications/. To use it, simply drop your folder of help files onto the tool. The Indexing Tools quickly scans all the pages and creates the search index file in the correct location.



drag-and-drop to enable searching

That's all there is to it. You can get fancier if you'd like, by adding keywords or changing the way some topics are indexed, but for most uses all you have to do is use the default Indexing Tools settings and drag-and-drop your help folder.

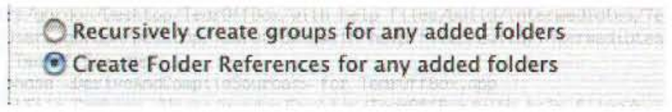
ADD YOUR HELP TO YOUR PROJECT

By adding the META tag and creating a search index you have finished everything you need to do to your help content. Next, you add the help files to your application's package and tell Cocoa you have a help system.

Importing into Project Builder

In Mac OS X help files are stored in an application's bundle. This keeps everything tidy – help always travels with the application when it's installed or moved on disk – and with Mac OS X's multi-lingual support it permits you to have localized help along with the application itself.

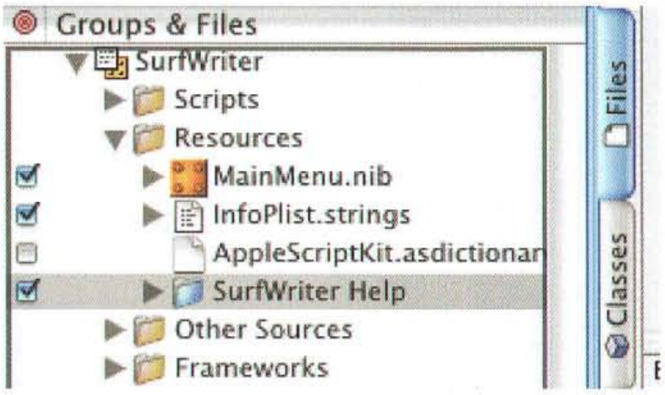
Select the Resources folder in Project Builder, and then choose Add Files from the Project menu. Select the main folder of your help content and turn on the option to create folder references, and then click the Add button.



Recursively create groups for any added folders
[✓ Create Folder References for any added folders]

Select this option when importing your help folder.

Your help folder is now listed in the Resources list.

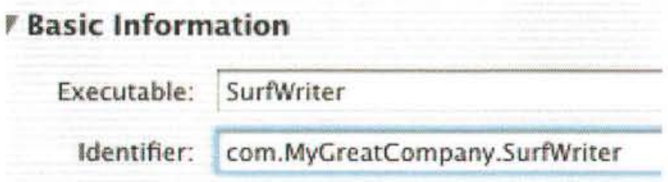


Help files added to a project.

Add Help to the Property List

Choose Edit Active Target from the Project menu. Then click the Application Settings tab to display the settings for your build.

In the Basic Information section, specify a bundle identifier. See *Inside Mac OS X: System Overview* for more detail, but in brief you specify a unique identifier for your application using Java-style naming.



Basic Information

Executable: SurfWriter

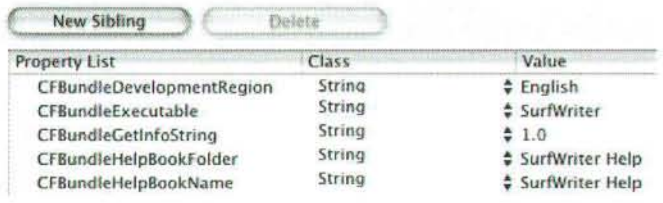
Identifier: com.MyGreatCompany.SurfWriter

Enter a bundle identifier

Click the Expert button in the top right corner of the settings pane. This allows you to edit the Property List directly.

Click the New Sibling button, then edit the new entry so the key is CFBundleHelpBookName and the value is a string that matches the contents of the AppleTitle tag that you put in your HTML.

Add another new sibling called CFBundleHelpBookFolder with the value being the name of your help folder that you previously imported. In our example both key values are the same because the folder where we store our help files is named the same as our book. That's not a requirement, but it is conventional.



Property List	Class	Value
CFBundleDevelopmentRegion	String	English
CFBundleExecutable	String	SurfWriter
CFBundleGetInfoString	String	1.0
CFBundleHelpBookFolder	String	SurfWriter Help
CFBundleHelpBookName	String	SurfWriter Help

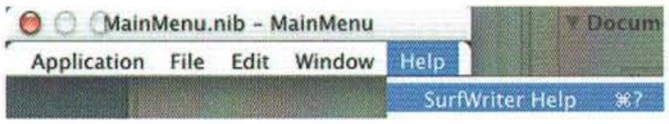
Specifying the help book and folder names

Edit The Help Menu

You're in the home stretch now. The last task is to rename the default Help menu that is automatically created for your application. You don't have to do anything to make the menu work; Cocoa provides the functionality automatically because of the changes you made to your property list.

In Project Builder, double-click your interface definition file (MainMenu.nib, unless you've changed it) to open it in Interface Builder.

In the menu layout, change the Help item to match the name of your help book. The Aqua guidelines recommend the form "*MyAppName Help*". Note that you automatically get the correct keyboard shortcut, courtesy of Cocoa.



Edit the Help menu item.

That's it, you're done! Build your app and try it out. When you select your new Help menu the Help Viewer will launch and open to your book. If it doesn't work make sure that the property list values match up with the appropriate AppleTitle and help folder names. Also check that you remembered to enter a bundle identifier for your application.

Your AppleScript Studio-based application is now one step closer to providing a great Aqua user experience and useful help for your customers. And, along the way, you've learned how to implement a help system for any Cocoa application.

Bibliography

Aqua Human Interface Guidelines

Help Chapter

- http://developer.apple.com/techpubs/macosx/Essentials/AquaHIGuidelines/AHIGHelp.fm/index.html

Providing User Assistance with Apple Help

- http://developer.apple.com/techpubs/macosx/Carbon/HumanInterfaceToolbox/AppleHelp/applehelp.html

Apple Help Reference

- http://developer.apple.com/techpubs/macosx/Carbon/HumanInterfaceToolbox/AppleHelp/Apple_Help/index.html

Apple Help WWDC session 125, Mac OS Track

Streaming video from Apple Developer Connection (Membership required. Log in, then choose View ADC TV)

- http://www.apple.com/developer/

Inside Mac OS X: System Overview

- http://developer.apple.com/techpubs/macosx/Essentials/SystemOverview/index.html

Felt Tip Software presents

Sound Studio 2.0

Built for Mac OS X

With Sound Studio 2.0, you can:

Digitize cassette tapes and vinyl records to make CDs
Record live performances and interviews
Record your favorite radio programs with Timer Recording
Edit out dead air, glitches, and mistakes in recording
Apply any of 19 effects including Reverb and Chorus
Create new sounds and loops

Sound Studio is an audio recording and editing application built for Mac OS X. It can also run on Mac OS versions 8.1 and up. Some systems may need a USB audio input device to record.

Only $49.99

Download 14-day trial version at

http://www.felttip.com/ss/

felt tip software

807 Keely Place, Philadelphia, PA 19128, USA

www.felttip.com

Building a LAN

Conceptually, building a LAN is very simple. Connect all your computers to a hub or switch, and then connect your internet connection to the same hub/switch. Everything works from there, and everybody is happy. This approach will work just fine for a low-traffic network with relatively few nodes, but as things begin to grow (and if all goes well they will grow) this approach can cause problems. It is best to introduce some structure to your LAN early in its life, so that growing is as pain-free as possible.

There are a few things that you should study before designing and building your LAN:

- The difference between half duplex and full duplex
- Basic information on the inner workings of Ethernet
- The difference between a hub and a switch
- Basic information about other layer 2 networks you will be using (802.11 wireless, for example).

We will cover these issues below.

Half Duplex vs. Full Duplex

A half duplex network is one where only one node may transmit at a given time. Standard ethernet networks have this property. The IEEE spec was updated to allow for full duplex operation, which means nodes may both transmit and receive at the same time. There is a restriction on this, which is that a full duplex segment may only have two nodes on it, or where the full duplex nodes are all connected to a switch.

How Ethernet works.

Ethernet is a layer 2, (originally) only half-duplex CSMA-CD network technology. Layer 2 describes where it falls in the OSI protocol stack (See the OSI Network Model sidebar). CSMA-CD means:

CS: Carrier Sense. This means that before sending any traffic on the network, any Ethernet node must listen on the network to sense if any traffic is already being transmitted. If it is, then the node must wait to send its traffic.

MA: Multi Access. An Ethernet network can have multiple nodes connected to it at the same time.

CD: Collision Detection. Due to the fact that network transmissions do not move across the network instantly, but with some latency it is possible that two nodes on a network might both detect at the same time that no traffic is being transmitted on the network, and thus start transmitting at the same time. When this happens, a collision is generated, which every node on the network hears. The nodes then use a random exponential backoff algorithm which defines how long the node must wait before transmitting again.

A collision can only occur while the ethernet header (preamble) is being transmitted. For Fast Ethernet this is actually the limiting factor in the length of a segment (this comes out to about 100 meters). For regular 10 Mbps ethernet the length of an ethernet segment is constrained by signal strength (this comes out to about 150 meters).

A collision domain is defined as a network that can only have one node transmitting at a time (or else a collision occurs).

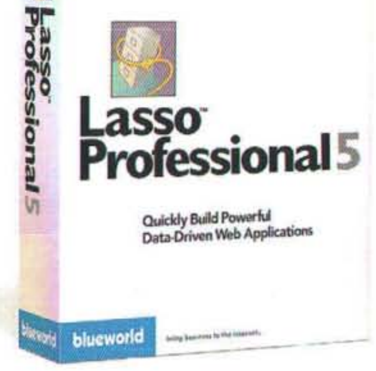
Now, depending on what sort of a core your ethernet network is comprised of you may have just one collision domain, or you may have several. There are two types of devices that can serve this role, hubs and switches. Hubs are far less expensive than switches (although as is always the case with computer equipment those prices are dropping). When a hub receives an ethernet frame on an interface, it indiscriminately forwards said frame to all of its ports. A hub has no knowledge of which devices are connected to its ports, so it relies on the device itself to decide which traffic is its own. This means that every port on the hub is a part of the same collision domain. A switch is essentially a more intelligent hub. Switches have knowledge about which MAC addresses are connected to which switch port. Thus, unicast frames are only forwarded to the port that has the destination host on it (broadcast packets are still forwarded to all of the switch ports). In a switch, each port represents a different collision domain. This means that multiple nodes connected to a switch

Alec Peterson is the Chief Technology Officer for Catbird Networks (http://www.catbird.com). Catbird provides performance and integrity services to businesses that have public-facing infrastructure and want to ensure that their Internet presence is always available and not defaced. Alec has extensive experience building and operating large networks, and is actively involved in Internet address allocation policy. He is currently chair of the American Registry for Internet Numbers Advisory Council.

"Awesome!"

"The power, quality and feature set of Lasso Professional 5 is very impressive. The new documentation is definitely among the best in the business."
Johan Sölve; Halmstad, Sweden

"Blue World has managed to add an immense amount of functionality and scalability without sacrificing any of the ease-of-use of previous versions. New features like LassoApps, custom tags and LassoScript create a development environment that seems almost limitless."
Tom Wiebe; Vancouver, Canada



"With Lasso, I have been able to single-handedly develop sophisticated solutions for internal and external use in less time than I see teams of people take with other middleware languages."
Greg Willits; Santa Ana, California

"This new release, I have got to admit, is truly amazing. The rich array of new features, the brand new documentation and the amazing new administration interface make Lasso Professional 5 definitely worthy of a purchase."
Brian Olsen; Brooklyn, New York

Upgrade today to the most powerful Web application server for Macintosh and beyond and you'll discover why so many Web developers claim Lasso Professional 5 is the must-have tool for quickly building and serving powerful data-driven Web sites.

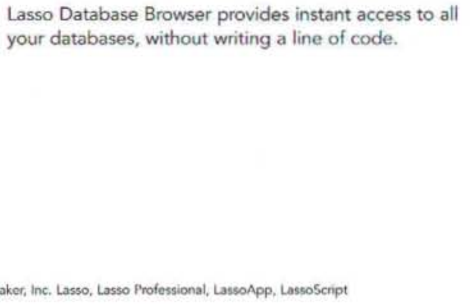
Lasso Professional 5 introduces a next-generation, object-oriented Web programming language, advanced Web application server administration, an embedded Lasso MySQL™ high-performance database server, a new distributed architecture, new platform and data source support, unprecedented extensibility and customization, 1800 pages of rewritten documentation and over 200 new features and enhancements. Lasso Professional 5 provides vast new power and features while maintaining the legendary ease-of-use, performance, and reliability that make Lasso the preferred tool for tens of thousands of Web developers, ISPs, and IT/IS professionals.

If you're serious about building and serving data-driven Web sites, but don't want to spend a serious amount of time getting it all to work, there's no better choice than Lasso Professional 5.

Visit the Blue World Web site today and see how Lasso products can help you quickly build and serve powerful data-driven Web sites.

Lasso – The Leading Web Tools for Macintosh and Beyond

Lasso Administration controls your entire setup via an attractive and intuitive Web browser interface.



Lasso Database Browser provides instant access to all your databases, without writing a line of code.

blueworld

www.blueworld.com

can be transmitting at the same time. However, if a switch port is operating in half-duplex mode, then the switch cannot transmit on a given port at the same time as another node on that same port. Note that you can connect multiple nodes to a single switch port by connecting a hub (or another switch) to a switch port, and you can also cascade switches (ie, connect switches to switches).

Collisions are a normal part of operation on a half-duplex ethernet network. However, the more active nodes you have on a network the more collisions you will have. Too many collisions can seriously degrade network performance. Based on this statement one might ask how many collisions is 'too many'. The best way to see if collisions are causing a problem is to ping across the network (ie, ping one host on the network from another one). Latency across the Ethernet should typically be only a couple of milliseconds, and packet loss should be 0. If either of these is not the case, then you probably have too many players on the network, and something should be done.

Troubleshooting tip: When a network is setup incorrectly, you may notice the 'late collision' counter incrementing on one or more of your ethernet devices. In a nutshell, this means that a collision occurred after the preamble of the ethernet frame. The most common cause of this is when you have one end of an ethernet connection running in half duplex mode, and the other end running in full duplex mode. In this case you will notice the late collision counter incrementing on the half duplex side, because the other side of the connection (which is full duplex) is expecting to be allowed to transmit and receive at the same time, hence it is ignoring collisions. A less common cause is when you have an ethernet network that is too long. Late collisions happen in this case because the beginning of the preamble has not reached the end of the network before the transmitting node has completed sending it, thus the transmitting node assumes that it is allowed to transmit the entire frame.

If your entire network is built using a single hub (or a series of hubs) then your first step should be to upgrade to a switch. This will reduce the number of devices on the same collision domain (by putting each node on its own collision domain). This in turn increases the number of nodes that can transmit at the same time (from 1 to n, where n is the number of active switch ports). Once you have a switch at the center of your network fabric you should not encounter many problems until you begin to run low on network ports. You can address this by either hanging hubs off of switch ports, or by stacking switches together. Many switches have their own proprietary trunking system for connecting them together. Others have gigabit ethernet ports for this purpose. Either method will work just fine.

Switches come in a variety of shapes and sizes. For example, you can get a managed switch, which allows you to hardwire ports to a specific duplex/speed setting among other things. Or you can get an unmanaged switch which theoretically takes care of this for you. If you want to run any full-duplex devices on your network the author strongly recommends getting a managed switch and hard-coding all ports that need to be full duplex to that setting. The protocol for auto-negotiating duplex very seldom works properly, even between devices that are made by the same vendor.

OTHER LAYER 2 PROTOCOLS

IEEE 802.11 is probably the most common non-ethernet LAN protocol being used today. Sometimes called wireless ethernet, IEEE 802.11 describes protocols that allow computers to communicate without physical wires between them. This protocol is a CSMA-CA protocol. You already know what the CSMA initials stand for. CA instead of CD means 'collision avoidance' instead of 'collision detection'. This means that on an 802.11 network collisions do not exist. Instead, the radios negotiate amongst themselves to decide who can speak at any given time.

OSI NETWORK STACK

Network protocols are built on top of one another. This is done using the OSI Network Model. Note that Internet Protocols are not designed by OSI, but rather by the IETF. Thus, some of the OSI layers are not cleanly represented by Internet Protocols. The layers are as follows:

Application layer: The application layer provides authentication and data syntax. Everything at this layer is application specific. HTTP is a well known application layer protocol.

Presentation layer: The presentation layer provides manipulation of application layer data, the most common way is through encryption. SSL is a presentation layer protocol.

Session layer: The session layer establishes, manages and terminates connections between applications.

Transport layer: The transport layer provides transparent data transfer between hosts. This layer is responsible for ensuring packet delivery and veracity as well as flow control. The combination of DNS and TCP or UDP perform the functions of the session layer and the transport layer.

Network layer: The network layer is responsible for routing and switching packets between nodes on the network. IP is a network layer protocol.

Data Link layer: The data link layer is responsible for assembling bits transmitted on a network into chunks of data, known as frames. Ethernet is a data link layer protocol.

Physical layer: The physical layer is responsible for representing electrical (or other) impulses as bits (1 or 0). Basically any physical media that has data transmitted over it is part of the physical layer.

The end result of this layering system is that higher level protocols do not need to have any knowledge of the structure or makeup of the underlying network. This is what allows the Internet to run over many different physical media (fiber optics, copper wires and air waves). When a web browser makes a connection to a web server, it has no idea how its traffic is routed, just that it got there.

802.11 relies on something referred to as an 'access point' for nodes to talk to one another as well as other networks (such as a physical ethernet network). The access point has two primary responsibilities:

1) Allow wireless nodes to talk to one another by acting as a central hub for all of the radios to register with.
2) Allow wireless nodes to connect to other types of layer 2 networks by acting as a bridge.

All traffic on a wireless network will go through an access point, regardless of whether it is destined for a terrestrial network or for another radio on the same wireless network. This isn't much of a problem since the wireless network is a single half-duplex segment anyway.

Scaling wireless networks has to take into account your goals. Scalability is accomplished by distributing multiple access points in an intelligent manner, based on your goals. You can use multiple access points in a given area because of the fact that 802.11 defines 13 channels that can all be used simultaneously on top of each other. If you intend to have a large number of radios in a small physical area, then you would want to have multiple access points in that same area, running on different channels. Should you need more than 13 access points, you can reuse channels as long as any two access points running on the same channel are not within range of each other. Access points distributed in this manner are typically all given their own ethernet connection to the LAN. It is possible using certain access points to have some radios act as repeaters (cisco/Aironet are known to the author to support this). This technique should only be used as a last resort to get some extra coverage, since it requires that the repeater radio run on the same channel as the one it connects to on the ethernet, thus it does not give you more wireless bandwidth.

Troubleshooting tip: If you are experiencing poor performance on your wireless network, do not automatically assume that the problem lies in the congestion of the wireless network itself. Remember that the access point is communicating using CSMA-CD to the ethernet network. So if it is receiving traffic from the ethernet and wireless networks at the same time, it needs to buffer data before it can be transmitted. Some access points only have the ability to speak half-duplex over their ethernet ports, and since they act as bridges they have implemented a very aggressive collision backoff algorithm (which means they give up trying to send a packet over the ethernet a lot more quickly than some other ethernet devices would). If you have this problem investigate whether you can get your access point to support full-duplex ethernet. Should that not work then you will want to distribute more access points on different channels to cover the same physical area.

However, if your scaling goals involve covering wide physical areas with relatively few wireless nodes, then you should look into getting a high gain omnidirectional antenna for your access point. This will significantly increase the range that a single access point can cover.

Since 802.11 has become so popular recently, there are now many vendors selling access points for over a large price range. The products that are billed as residential products can certainly be used in a commercial environment, however you do get what you pay for. Read the fine print when purchasing such devices, as some of them have a very low limit on the number of radios that may be connected to the access point at a given time. The best advice is to evaluate your needs and then make sure you know everything you need to know about a product before purchasing it. Note that should you want to setup a repeater radio somewhere in your network you will probably need to purchase a high-end access point for both the repeater radio as well as the one connected to the physical network.

It is also possible to use 802.11 for point to point bridging, by using directional antennas. However, such implementations are beyond the scope of this article.

Conclusion

As with most things in life, LANs get more complicated the larger they get. If you have a specific network design in mind when getting started on your LAN it will make growing it a much less painful task

DO YOU NEED TO "MODERNIZE" *YOUR* SEARCH TECHNOLOGY?

POWER TOOLS FOR YOUR WEBSITE:

Onix - Search and Retrieval Engine
RouteX - Document Filtering and Routing Engine
Brevity - Document Summarization Engine

http://www.lextek.com

THE MISSING LINK!

Don't throw out your old machine or peripheral, just find the right adapter!

K3 Smart Card Reader for USB

(with optional developer kit – let your App read Smart Cards!)

$184.95

2nd Wave FireWire to SCSI Converter

Connect SCSI Devices to your FireWire Mac!

$78.95

DevDepot your source for connectors and adapters of all kinds!

DEV DEPOT℠

www.devdepot.com/connectors

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)

Belkin USB & FireWire Combo PCI Card

Get two for one, add the power of FireWire and the flexibility of USB to your PCI Mac with one card!

$79.95

DVIator

Connect an Apple flat panel display to your DVI Mac!

$147.95

KEYSPAN Twin Serial

Turn one USB port into two serial ports!

$69.95

Griffin iMate

Connect ADB devices to your USB Mac!

$34.95

Griffin iMic

Add audio in and out to your USB Mac!

$32.95

Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • E-mail: orders@devdepot.com

By Rich Morin

The BSD Man Pages

How to find them; how to read them.

Welcome

"Section 7", the name of this column, is also the shorthand term for the "Miscellaneous Information" section of the Berkeley Software Distribution's manual pages (known to their friends as the BSD "man pages"). This column will feature miscellaneous information on BSD-related aspects of Mac OS X, including application design, documentation, and even the politics of the developer and user communities.

Mac OS X is the most widely-disseminated version of BSD, but many Mac developers are a bit unclear on what BSD actually is. So, a word of explanation may be in order. BSD is a well-respected operating system, with a long history of solid engineering. Derived from Unix 32V, BSD is used for network servers, firewalls, and many other mission-critical applications.

Darwin (and thereby Mac OS X) is based on the FreeBSD and NetBSD distributions. As a result, it can take advantage of a well-developed operating system environment and thousands of "ported" applications from the FreeBSD Ports Collection. Clearly, BSD's solid technology and active developer community make a powerful contribution to the Mac OS X story.

Getting Started

BSD (and other Open Source) developers maintain several thousand "man pages" (manual pages), covering a wide range of topics. Because the pages are written by developers, they tend to be authoritative, consistently organized, current, detailed, and refreshingly honest. On the other hand, they also tend to be terse, tightly focused, and a bit unpolished in spots.

Consequently, reading man pages may not be the best way to approach a new and complex topic (e.g., NFS, socket programming), but it is an extremely handy way to learn about specific commands, remind yourself about forgotten details, etc. Once you've read a few man pages, I predict that you'll get addicted to their strong points; you may even (eventually) forgive their deficiencies.

Although it isn't critically necessary, you'll find it useful (and fun!) to have some Terminal windows on hand while you read this article. For one thing, you'll be able to see the actual output of the commands I'm describing; for another, you'll be able to try things out.

Using the Finder, navigate through the Applications and Utilities folders. When you find the icon for the Terminal Utility, drag it into the Dock. Now, double-click the icon twice, creating two Terminal windows. Unlike typical applications, these windows are not closely tied to each other. In fact, each Terminal window provides an independent BSD "session".

You should probably spend a few seconds arranging the windows, so that you can see them both at the same time. Also, stretch at least one of them vertically (keeping the width constant). This will let you see more of your commands' output without scrolling.

To view a particular man page, you must ask the system to run the "man" command, giving it the name of the page you want to see. We can try this out by asking for the man page that describes the "null" device:

```
[localhost:~] rdm% man null
NULL(4)          System Programmer's Manual              NULL(4)

NAME
     null - the null device

DESCRIPTION
     The null device accepts and reads data as any ordinary
     (and willing) file - but throws it away. The length of
     the null device is always zero.

FILES
     /dev/null

HISTORY
     A null device appeared in Version 7 AT&T UNIX.
```

Headings

As the example shows, each part of the man page is introduced by a descriptive heading. Some headings (e.g.,

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1985 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.

NAME, DESCRIPTION) will show up on every man page; others (e.g., FILES, HISTORY) only appear when needed. Some, finally, are invented by the page's author. Here are some common headings, in the order they would normally appear on a man page:

- **NAME** The following text gives the name(s) of the described item(s), followed by a terse description. In the case of commands, only one name will typically be given. In the case of library functions, however, there may be a long list of entry points.
- **SYNOPSIS** The following text gives a (stylized) description of the normal usage mode for the item(s). In the case of commands, this will show the possible option flags; in the case of system calls and library functions, it will show representative declarations, etc.
- **DESCRIPTION** The following text gives an extended description of the item(s), including options, usage considerations, etc. Some descriptions go on at great length.
- **RETURN VALUES** The following text describes the return values for the item(s).
- **ERRORS** The following text lists possible error conditions and resulting actions.
- **FILES** The following text lists related files.
- **SEE ALSO** The following text lists related man pages.
- **STANDARDS** The following text lists applicable standards, sometimes discussing the degree to which the described item(s) comply.
- **BUGS** The following text lists ways in which the command fails to meet the documentor's expectations. Some vendors find this section embarassing, so they rename or remove it. Feh!
- **NOTES** The following text lists ancillary information which is not appropriate to any of the other sections.
- **HISTORY** The following text tells where (i.e., in which BSD or Unix variant) the item first appeared, etc.

PAGING

With all of this detail, some man pages can (and do :) go on for quite a while. In fact, multiple-page "man pages" are quite common. So, the man command is set up to let the reader "page" through the text. The default "pager" on Mac OS X is "more"; for a rundown on its capabilities, look at its man page!

```
[localhost:~] rdm% man more
```

Although more isn't very sophisticated, it does respond to the space bar (skip to the next page), the return key (move forward one line), "q" (quit), and a few other commands. If you want more features, try the GNU Project's "less" command (less is more, more or less):

```
[localhost:~] rdm% setenv PAGER less
```

http://www.scientific.com

Professional Software Developers

Looking for career opportunities?
Check out our website!
Nationwide Service
Employment Assistance
Resume Help
Marketability Assessment
Never a fee

Scientific Placement, Inc.

800-231-5920 800-757-9003 (Fax)
das@scientific.com

high quality - competitive rates - 16 years experience - award winning

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

Full Spectrum Software
Development & Testing

Device Drivers
Porting **Plug-ins**
TCP/IP
Carbon / OSX
Cross Platform Development

One Bridge Street
Newton, MA 02458
617-965-0029
www.FullSpectrumSoftware.com

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

competitive rates - 16 years experience - award winning - high quality

Sections

The man pages cover a wide range of topics: administrative and user commands, C language interfaces, devices, etc. If all of the topics were simply sorted together, there would be name clashes. For instance, there is a passwd file, as well as a passwd command. To keep things straight, the man pages are divided into nine sections:

- Section 1 (General Commands). These are commands that any user might enter from the command line or use in a "shell script". Some example commands include cd, ls, and rm.
- Section 2 (System Calls). These are C language interfaces to kernel-provided services. Note that a system call, by definition, causes a context switch into the kernel. The application is halted until the system call completes. Some example system calls include close and open.
- Section 3 (Library Functions). These are also C language interfaces, but the function runs (in general) with the application's context. Of course, some library functions make system calls, which do context switches, as described above. Some example library functions include fopen, scanf, and sqrt.
- Section 4 (Devices and Device Drivers). These are the external interfaces to system devices. Some devices (e.g., /dev/null, /dev/tty) are intended for direct (command-line) use; others are only accessed by application or system-level code. Consequently, the descriptions in this section vary considerably in level and coverage.
- Section 5 (File Formats). Many BSD activities are controlled or monitored by means of files. This section describes some of the critical files that an administrator might need to read or edit. Note, however, that Mac OS X uses different administrative mechanisms than most other BSD variants, so your mileage will definitely vary!
- Section 6 (Games). Some BSD distributions include recreational programs; others do not. By placing these programs in a special area, the operating system eases monitoring and control of their use.
- Section 7 (Miscellaneous Information). This section includes descriptions that are not tied to any given command, file, or function. Look here for high-level introductions, etc.
- Section 8 (System Maintenance and Operation Commands). These are commands that the system administrator might enter from the command line or use in a "shell script". Some example commands include chown, fsck, and tunefs.
- Section 9 (System Kernel Interfaces). These are C language interfaces to kernel-provided services. They differ from system calls in that they can only be called from inside the kernel. Thus, they are only of interest to kernel programmers. Mac OS X does not currently have this section, but FreeBSD does, so I expect it to show up eventually.

Navigation

Going back to the case of passwd, we see that the passwd command would be described in Section 1, while the passwd file would be described in Section 5. The specific man pages would normally be referred to as "passwd(1)" and "passwd(5)". By default, the man command will display the first page it finds, going in ascending order through the sections. You can make sure that you get passwd(5), however, by typing "man 5 passwd". Try it!

Unlike the hierarchical "help systems", as found on Cicso routers or operating systems such as VMS, man pages have no automated support for navigation. Some experimental variants use HTML to display man pages, turning SEE ALSO entries into hyperlinks. Most man systems, however, require that you do your own navigation.

Part of the reason for this lies in the sheer scale of the BSD command set. There are several hundred commands and thousands of functions. Trying to place them all in a meaningful hierarchy (or even defensible topical groupings) is not a trivial task! Nonetheless, some help is available.

To find a man page when you don't know the exact name, use apropos(1). It displays all the "NAME" lines that contain the specified text strings. If you are trying for a text string that contains non-alphanumeric characters, be sure to enclose the string in quotes. Otherwise, you'll get all the pages that match any "word" (i.e., sequence of non-blank characters) in the string. Here are some interesting uses of the apropos command; try them out!

```
[localhost:~] rdm% apropos device
[localhost:~] rdm% apropos dev
[localhost:~] rdm% apropos '(4)'
[localhost:~] rdm% apropos 'file system'
[localhost:~] rdm% apropos file system
```

Although BSD provides several hundred commands, only a few are needed for everyday use. Next month, we'll look at some basic BSD commands, as well as a smattering of shell (command line interface) syntax.

Without this piece, you're not done.

It's not enough just to write solid code anymore. Users expect an easy-to-use installer when they buy your software.

InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.

Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers. Even create demoware in minutes.

Visit **http://www.stuffit.com/installermaker/** to also learn about creating Mac® OS X compatible installers with InstallerMaker.

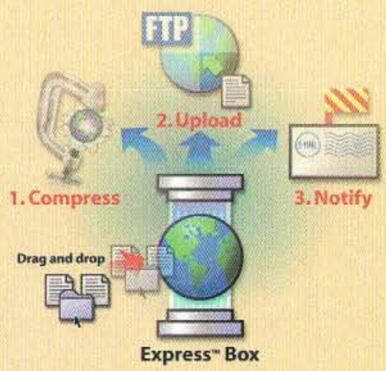
Then you're done. **StuffIt InstallerMaker**

The complete installation and distribution solution.℠

Get your beta testing started faster.

Guarantee your builds arrive safe and sound! New **StuffIt Express**™ automates file compression and transfer tasks to eliminate costly errors every time you exchange data. By creating custom drop box applications, you and those you communicate with can accomplish complex file transfer tasks with a simple drag and drop. Choose from up to 26 file transfer steps with no coding required. Boost productivity and save time by distributing your custom Express boxes to everyone on your list.

http://www.stuffit.com/express/enterprisemac.



by Tim Monroe

Big

Playing QuickTime Movies Fullscreen

Introduction

It's sometimes remarked that when QuickTime was first introduced, it was able to play movies that were only about the size of a postage stamp. It turns out that this isn't quite true. The very first QuickTime CDs distributed to software developers contained a number of sample movies that are 320 by 240 pixels, or just under 4.5 by 3.5 inches (which is significantly larger than any postage stamp I've ever seen). What *is* true is that certain kinds of movies — in particular, full-motion movies encoded using the video compressor — had to be kept small in order to achieve a reasonable frame rate on playback. These kinds of movies typically had to be somewhere in the order of 160 by 120 pixels to get a playback rate of about 12 frames per second. Not great, but pretty good for a software-only movie playback system on a Mac II in 1991.

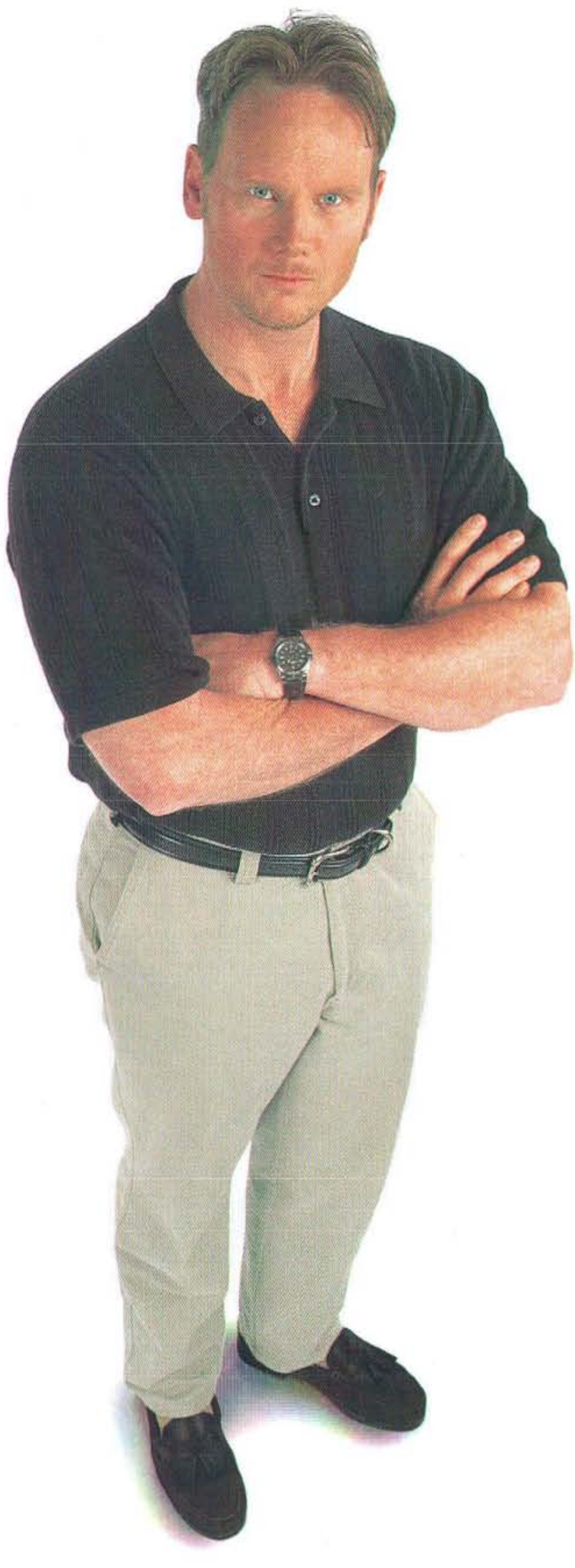
Since then, computer hardware and software technologies have advanced to the point that QuickTime can achieve smooth playback for much larger movies, at frames rates of 24 to 30 frames per second (or even greater). From QuickTime version 2.1 onward, it has also been possible to play QuickTime movies back *fullscreen*, so that the movie occupies an entire screen. Figure 1 shows a frame of a movie being played back fullscreen; as you can see, there is no menu bar and the window that contains the movie does not have a window frame or title bar. In addition, the control strip (on classic Mac operating systems), the dock (on Mac OS X), or the taskbar (on Windows operating systems) is hidden while a movie is playing fullscreen. At its natural proportions, this movie does not completely fill the screen, so it's centered horizontally with the edges drawn in black.

***Figure 1**: A QuickTime movie played fullscreen*

Fullscreen movie playback is fairly common in games, especially for the cut-scenes that occur between game levels. It has also recently become popular for many of the movie trailers posted to the web in QuickTime format.

In this article, we're going to learn how to play QuickTime movies fullscreen. I mentioned in an earlier article ("The Flash", in *MacTech*, January 2002) that fullscreen movie playback is accomplished primarily using the two functions BeginFullScreen and EndFullScreen. When using these functions to integrate fullscreen playback into our sample applications, however, we'll need to pay attention to a number of issues, including saving and restoring the state of our movie windows and their associated movie controllers. We also want to take a look at the wired action introduced in QuickTime 5.0.1 that a movie can use to begin and end fullscreen playback. Along the way, we'll touch on a few topics of general interest to QuickTime developers, including time base callback functions.

Tim Monroe in a member of the QuickTime engineering team. You can contact him at monroe@apple.com. The views expressed here are not necessarily shared by his employer.



My files are important.

Very important.

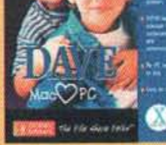
I only share my files with those I trust.

I trust DAVE.

Mac to PC, PC to Mac. Cross-platform file and print sharing is too vital to your business to risk. Trust Thursby, the company with 15 years experience. Trust DAVE, the solution with a proven track record. Share files and printers across a network with no barriers. DAVE installs on your Mac with no additional software required for the PC. It's fast, secure and easy to use. Download a free evaluation today!

Trust DAVE®.

New! Support for inkjet printing



Macworld ●●●●½

www.thursby.com

THURSBY Software

The File and print Share Folks™

Built for Mac OS X

Our sample application this month is called QTBigScreen. As usual, it's based on the QTShell sample application and adds support for entering and exiting fullscreen mode. The Test menu of QTBigScreen is shown in **Figure 2**; it contains just one item, which allows the user to put the frontmost movie window into fullscreen mode.

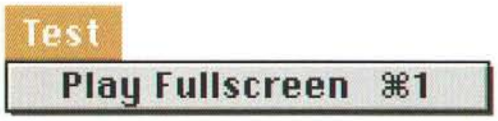


***Figure 2**: The Test menu of QTBigScreen*

There is no menu item for exiting fullscreen mode; instead, we'll follow the example of QuickTime Player and return to the normal windowed mode when the user types the Escape key or when a non-interactive and non-looped movie reaches the end.

THE THEORY

Let's begin by taking a look at the BeginFullScreen and EndFullScreen functions. As I mentioned, things tend to get a tad lengthy when we use these calls in a real-life application, so it's good to have a firm grasp on how they work before we try to do that. In this article, we are going to focus on using these functions to play QuickTime movies fullscreen, but they can in fact be used to display *any* kind of content in a fullscreen window. Moreover, we can use these functions simply to change the resolution of a screen, without wanting to take over the entire screen. In short, there's a lot going on with these two functions that we need to understand clearly before we attempt to use them in our applications.

Entering and Exiting Fullscreen Mode

The BeginFullScreen function is declared essentially like this:

```
OSErr BeginFullScreen (
        Ptr *restoreState,
        GDHandle whichGD,
        short *desiredWidth,
        short *desiredHeight,
        WindowRef *newWindow,
        RGBColor *eraseColor,
        long flags);
```

The key input parameters are desiredWidth and desiredHeight, which are pointers to the width and height of the movie (or image, or other content) that we want to display fullscreen. BeginFullScreen creates a new window that is at least that large and returns a window pointer for that window in the location pointed to by the newWindow parameter. BeginFullScreen also erases the screen (using the color specified by the eraseColor parameter) and (depending on the value of the flags parameter, which we'll consider in a moment) hides the menu bar and control strip. The whichGD parameter indicates which graphics device we want to put into fullscreen mode; we shall always pass the value NULL to select the main screen.

BeginFullScreen also returns, through the restoreState parameter, a pointer to a block of memory that contains information on how to return from fullscreen mode to normal windowed mode. We exit fullscreen mode by passing that pointer to EndFullScreen, which is declared like this:

```
OSErr EndFullScreen (Ptr fullState, long flags);
```

Note that the restoreState (or fullState) pointer is opaque and is owned by QuickTime; we shouldn't do anything with it except pass it to EndFullScreen when we are ready to exit fullscreen mode. Similarly, the newWindow window pointer is owned by QuickTime, which will dispose of it after we call EndFullScreen.

The flags parameter passed to EndFullScreen is unused and should be set to 0. The flags parameter passed to BeginFullScreen controls several aspects of fullscreen mode. Currently we can use these constants to set the value of this parameter:

```
enum {
  fullScreenHideCursor          = 1L << 0,
  fullScreenAllowEvents         = 1L << 1,
  fullScreenDontChangeMenuBar   = 1L << 2,
  fullScreenPreflightSize       = 1L << 3
};
```

The fullScreenHideCursor flag indicates that BeginFullScreen should hide the cursor while the screen is in fullscreen mode. This is useful if we just want to play a movie fullscreen from start to finish, but it's less useful for interactive movies played fullscreen. In QTBigScreen, we will not set this flag when we call BeginFullScreen. The fullScreenAllowEvents flag indicates that our application intends to allow other open applications to receive processing time; in general, we should set this flag. The fullScreenDontChangeMenuBar flag indicates that BeginFullScreen should not hide the menu bar.

The fullScreenPreflightSize flag is of particular interest. If we set this flag, then BeginFullScreen does not change any screen settings and does not return a new window to us. Instead, it returns, through the desiredWidth and desiredHeight parameters, the width and height that the screen would have been set to if the fullScreenPreflightSize flag had not been set. We can use that flag to determine the size of the fullscreen window without actually entering fullscreen mode.

Changing the Screen Resolution

Why is this useful? Recall that BeginFullScreen will create a window that is at least as large as the height and width we pass it. In addition, BeginFullScreen will change the screen resolution to the closest resolution that contains that window. If, for instance, the main screen is originally set to a resolution of 1024 by 768 and if we pass a width and height of (say) 762 and 560, then BeginFullScreen will change the resolution of the screen to 800 by 600 (assuming that the monitor supports that resolution).

We can exploit this behavior in several ways. First, we can use BeginFullScreen to determine the current screen resolution. If we set 0 to be the value pointed to by both the desiredWidth and desiredHeight parameters, then BeginFullScreen leaves the dimensions of the screen unchanged but returns to us the current dimensions of the screen in the locations pointed to by those parameters. If we also pass the fullScreenPreflightSize flag, then BeginFullScreen doesn't change any of the current screen settings. The end result is that we are given the current height and width of the screen (that is, its resolution). Listing 1 defines the function QTUtils_GetScreenResolution, which retrieves the current resolution of the main screen.

Listing 1: Getting the current resolution of the main screen

QTUtils_GetScreenResolution

```
OSErr QTUtils_GetScreenResolution
        (short *thePixelsHoriz, short *thePixelsVert)
{
  Ptr       myDummyPtr = NULL;
  OSErr     myErr = noErr;

  if ((thePixelsHoriz == NULL) || (thePixelsVert == NULL))
    return(paramErr);

  *thePixelsHoriz = 0;
  *thePixelsVert = 0;

  myErr = BeginFullScreen(&myDummyPtr, NULL, thePixelsHoriz,
        thePixelsVert, NULL, NULL, fullScreenPreflightSize);

  return(myErr);
}
```

Notice that we need to pass the address of a variable of type Ptr as the first parameter to BeginFullScreen, even though it does not return an actual pointer in that location. If we look at the value of myDummyPtr after calling BeginFullScreen here, we'll see that it's still NULL. So there is no need to call EndFullScreen to dispose of that pointer.

We can also use BeginFullScreen to change the screen resolution without hiding the menu bar. We simply pass in the desired height and width, and we set the flags parameter so that the menu bar is not hidden. In this case, we need to pass the value NULL for the newWindow parameter, indicating that we don't want a new window to be created. Here's how we could set the main screen to a resolution of 800 by 600:

```
 myHorizPixels = 800;
myVertPixels = 600;

myErr = BeginFullScreen(&gRestoreState, NULL,
         &myHorizPixels, &myVertPixels, NULL, NULL,
         fullScreenDontChangeMenuBar);
```

We need to keep track of the restore state so that we can undo the resolution change at some future time. You should keep in mind that there is no way to suppress the hiding of

Fetch 4.0

Walk the dog.

X

Fetchsoftworks.com

More Features • Fewer Bugs

Bigger Icon

the control strip (or dock or taskbar) when you call **BeginFullScreen**. If you want the control strip (or dock or taskbar) to remain visible after you adjust the resolution, you'll need to programmatically reshow it. (For instance, on Windows you could call the QTML function **ShowHideTaskBar** to reshow the taskbar.) It would be nice if QuickTime defined a flag that would allow us to request that the control strip (or its ilk) remain visible after a call to **BeginFullScreen**.

Scaling the Movie

In QTBigScreen, we don't want the resolution of the main screen to be changed when we play a movie fullscreen. We can accomplish this by scaling the movie so that the requested size is large enough to occupy all or almost all of the main screen at its current resolution. So we're going to end up calling **BeginFullScreen** twice, once to get the current resolution (as we did in Listing 1) and again to put a movie window into fullscreen mode. But before we call **BeginFullScreen** the second time, we need to do a little mathematics to figure out how to scale the movie so that it retains its original aspect ratio but fills as much of the screen as possible.

We begin by calling **BeginFullScreen** to get the current resolution of the screen:

```
 short        myScreenWidth = 0;
short         myScreenHeight = 0;

myErr = BeginFullScreen(&(**myAppData).fRestoreState, NULL,
          &myScreenWidth, &myScreenHeight, NULL, NULL,
          fullScreenPreflightSize);
```

As you can see, we are storing the restore state in the application data record. Ultimately we're going to have to maintain about a dozen pieces of data in that record (which we'll encounter shortly). Once we've retrieved the current resolution, let's make a copy of that information:

```
myOrigScreenHeight = myScreenHeight;
myOrigScreenWidth = myScreenWidth;
```

Now we need to get the natural size of the movie we want to play fullscreen. We can do this by calling **GetMovieNaturalBoundsRect**:

```
 GetMovieNaturalBoundsRect(myMovie, &myRect);
MacOffsetRect(&myRect, -myRect.left, -myRect.top);

myMovieWidth = myRect.right;
myMovieHeight = myRect.bottom;
```

(Calling **GetMovieBox** here wouldn't work quite right, since the user might have resized the movie window before putting it into fullscreen mode.)

And now we can calculate the aspect ratios of the screen and the movie:

```
myMovieRatio = FixRatio(myMovieWidth, myMovieHeight);
myScreenRatio = FixRatio(myScreenWidth, myScreenHeight);
```

We use these ratios to determine which dimension of the movie should be scaled to completely fill the corresponding dimension of the screen. The math required is simple:

```
if (myMovieRatio > myScreenRatio) {
  myMovieHeight = (myScreenWidth * myMovieHeight) /
                                             myMovieWidth;
  myMovieWidth = myScreenWidth;
} else {
  myMovieWidth = (myScreenHeight * myMovieWidth) /
                                             myMovieHeight;
  myMovieHeight = myScreenHeight;
}
```

At this point, we know the desired size of the movie and

Valentina

Object-Relational SQL Database

The fastest database engine for MacOS/Windows

It operates 100's and sometimes a 1000 times faster than other systems

www.paradigmasoft.com

Hosted by macserve.net
Download full featured evaluation version

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

EDITORS' CHOICE AWARD 1993 MACUSER WINNER

Accelerated for Power Macintosh

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater, 1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

hence we can call BeginFullScreen once again:

```
myScreenWidth = myMovieWidth;
myScreenHeight = myMovieHeight;
myErr = BeginFullScreen(&(**myAppData).fRestoreState, NULL,
         &myScreenWidth, &myScreenHeight,
         &(**myAppData).fFullScreenWindow, &myColor,
         fullScreenAllowEvents);
```

If BeginFullScreen returns successfully, then (**myAppData).fFullScreenWindow contains a window pointer to the new fullscreen window and myScreenWidth and myScreenHeight point to the actual width and height of that window. If the aspect ratio of the movie does not exactly match that of the screen, then we need to move the movie down or to the right so that it is centered in the fullscreen window (see **Figure 1** again). First we set the movie's rectangle:

```
MacSetRect(&myRect, 0, 0, myMovieWidth, myMovieHeight);
```

And then we nudge the movie down or to the right to center it on the screen:

```
MacOffsetRect(&myRect, (myScreenWidth - myMovieWidth) / 2,
         (myScreenHeight - myMovieHeight) / 2);
SetMovieBox(myMovie, &myRect);
```

There is one final "gotcha" we need to watch out for. Although we scaled the movie up so that one of its dimensions extends for the full width or height of the screen, it's possible that the expanded movie size exactly matches a screen resolution that is not the same as the original screen resolution. Suppose, for instance, that our movie has a natural size of 480 by 360 pixels. This movie, when scaled up, will exactly fill a screen that is 1024 by 768. However, the "megawide" screen on the Titanium PowerBook has a natural resolution of 1152 by 768, and it also supports the resolution of 1024 by 768 (by blanking 64 pixels on the left and right sides of the screen). If we play this movie fullscreen on that PowerBook, our existing code will result in the screen resolution being changed to 1024 by 768; in that case, there is no need to nudge the movie down or to the right.

So before we adjust the movie's rectangle, we should check to see whether the screen resolution did in fact change. We cleverly saved the original screen resolution, and our second call to BeginFullScreen gives us back the current screen resolution; we can compare the current with the original and then nudge the movie only if the resolution has not changed:

```
if ((myScreenWidth == myOrigScreenWidth) &&
         (myScreenHeight == myOrigScreenHeight))
   MacOffsetRect(&myRect, (myScreenWidth - myMovieWidth) / 
2,
         (myScreenHeight - myMovieHeight) / 2);
```

Once again, it might be nice if BeginFullScreen supported a flag that instructed it not to change the screen resolution.

The Practice

So, we now understand how to use BeginFullScreen and EndFullScreen to enter and exit fullscreen mode. We've seen how to request a fullscreen window size that makes our movie as large as possible while preserving its original aspect ratio and also preventing changes in the screen resolution (whenever possible). And we've seen how to adjust the movie box — the rectangle in which the movie is drawn inside its window — so that the movie is nicely centered in the fullscreen window. Aren't we done yet?

No. There are still a couple of issues we need to address. Our basic sample application framework was not developed with the intention of supporting fullscreen movie playback, so a couple of our framework functions need some minor tweaking. More importantly, we need to keep track of some information (including the restore information and the new window pointer) for each movie window we put into fullscreen mode. And of course, we need to make sure that events get passed to a movie playing fullscreen. In this section we'll tackle these issues.

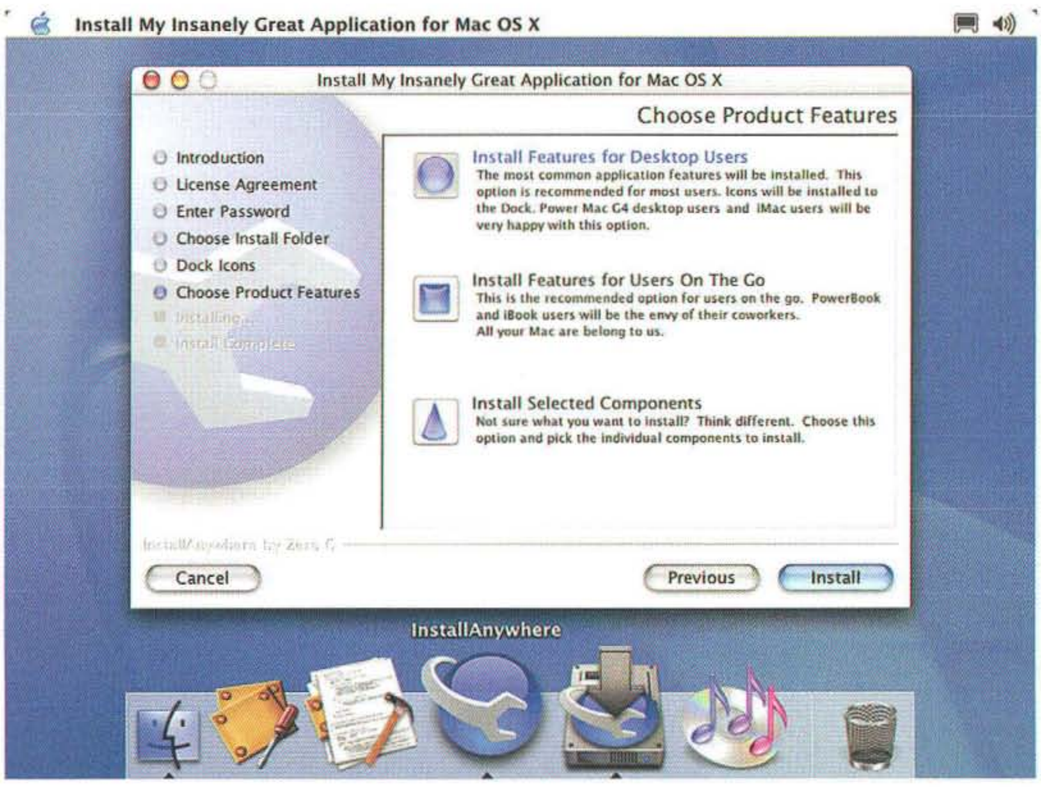
Initializing the Movie Window Data

Ideally, we'd like the user to be able to put any of our application's movie windows into fullscreen mode and then back into normal mode. We therefore need to keep track of the restore state information and the new window pointer returned by BeginFullScreen, as well as a few other pieces of information. We'll define a custom application data record, like this:

```
typedef struct ApplicationDataRecord {
   WindowReference     fOrigWindow;
   WindowPtr           fFullScreenWindow;
   Ptr                 fRestoreState;
   GWorldPtr           fOrigMovieGWorld;
   Rect                fOrigMovieRect;
   Rect                fOrigControllerRect;
   Boolean             fOrigControllerVis;
   Boolean             fOrigControllerAttached;
   QTCallBack          fCallBack;
   QTCallBackUPP       fCallBackUPP;
   Boolean             fEndFullscreenNeeded;
   Boolean             fDestroyWindowNeeded;
} ApplicationDataRecord, *ApplicationDataPtr,
         **ApplicationDataHdl;
```

The fOrigWindow field will contain the original window reference — that is, a reference to the window that the user puts into fullscreen mode. (Remember that the actual type of this object varies; on Macintosh systems, it's of type WindowPtr, while on Windows systems it's of type HWND.) The fRestoreState and fFullScreenWindow fields hold the restore state and window pointer returned by BeginFullScreen. The next five fields hold information about the state of the movie window at the time it's put into fullscreen mode; for instance, the fOrigMovieRect field holds the movie rectangle, and fOrigControllerVis indicates whether the controller bar was visible when we called BeginFullScreen. We'll need these pieces of information when we restore the movie to its normal windowed state.

LOOKING GOOD



Install My Insanely Great Application for Mac OS X

Choose Product Features

Install Features for Desktop Users
The most common application features will be installed. This option is recommended for most users. Icons will be installed to the Dock. Power Mac G4 desktop users and iMac users will be very happy with this option.

Install Features for Users On The Go
This is the recommended option for users on the go. PowerBook and iBook users will be the envy of their coworkers.
All your Mac are belong to us.

Install Selected Components
Not sure what you want to install? Think different. Choose this option and pick the individual components to install.

Looking good on Mac OS X is easy with InstallAnywhere!

Let's face it, first impressions are lasting ones. You don't build beautiful applications for Mac OS X only to deliver them with yesterday's tired installers.

Zero G has created a powerful and intuitive installation and configuration solution that is true to today's Aqua look and feel.

InstallAnywhere has complete support for Mac OS X, such as authentication, file permissions, and platform specific locations including installing icons into the Dock, making it the only installer you'll ever need.

And when you're ready to deploy to Mac OS Classic, Linux, Solaris, AIX, HP-UX, or even <gasp!> Windows, InstallAnywhere is ready too - anytime, anywhere, on any platform.

Software innovators like Adobe, Apple, BEA, Borland, HP, i2, Iona, Sun, TIBCO and Xerox already depend on InstallAnywhere for their software deployment needs.

See for yourself why InstallAnywhere, the world's leading multi-platform deployment solution, is also the best looking and most powerful installer for Mac OS X.

InstallAnywhere

zero G
ZERO G

Download a free trial version from http://www.ZeroG.com/goto/macosx

Built for Mac OS X

The fCallBack and fCallBackUPP fields hold information related to a time base callback function; we use this function to automatically return a movie from fullscreen state to normal state when the movie reaches the end, to mimic the behavior of QuickTime Player. (I personally don't like this behavior, but it's useful to know how to implement it. I've conditionalized the callback code using the compiler macro END_FULLSCREEN_AT_MOVIE_END, so it's easy enough to turn off.)

The last two fields of our application data structure are Boolean values that indicate whether the associated movie window should be returned from fullscreen mode to normal mode and whether the associated movie window should be closed. Later we'll see why we need these fields.

I'm also going to introduce a global variable that holds the window object for the window that is currently in fullscreen mode:

```
WindowObject          gFullScreenWindowObject = NULL;
```

It would be possible to determine which, if any, movie window is in fullscreen mode by iterating through all open movie windows and checking their application data record (to see if fFullScreenWindow is non-NULL), but using a global variable will simplify our code.

When we first open a movie window, we shall call QTBig_InitWindowData (Listing 2) to create this custom application data record and initialize its fields.

Listing 2: Initializing application-specific window data

QTBig_InitWindowData

```
ApplicationDataHdl QTBig_InitWindowData
        (WindowObject theWindowObject)
{
  ApplicationDataHdl    myAppData = NULL;

  // if we already have some window data, dump it
  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(theWindowObject);
  if (myAppData != NULL)
    QTBig_DumpWindowData(theWindowObject);

  // allocate a new application data handle
  myAppData = (ApplicationDataHdl)
        NewHandleClear(sizeof(ApplicationDataRecord));

  return(myAppData);
}
```

And when we close a movie window, we want to deallocate the application data record. For this, we call the function QTBig_DumpWindowData, defined in Listing 3. Notice that we first check to see whether the window is in fullscreen mode; if it is, we call our function QTBig_StopFullscreen to return it to normal mode.

Listing 3: Destroying application-specific window data

QTBig_DumpWindowData

```
void QTBig_DumpWindowData (WindowObject theWindowObject)
{
  ApplicationDataHdl  myAppData = NULL;

  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(theWindowObject);
  if (myAppData != NULL) {
    if ((**myAppData).fFullScreenWindow != NULL)
      QTBig_StopFullscreen(theWindowObject);

    DisposeHandle((Handle)myAppData);
    (**theWindowObject).fAppData = NULL;
  }
}
```

Entering Fullscreen Mode

Let's revisit for a moment our basic scheme for opening movie windows and keeping track of window-specific data. On Macintosh systems, we call NewCWindow to obtain a window pointer to a new window. We allocate a new window object record and store a handle to that record as the window reference constant (by calling SetWRefCon). This allows us to retrieve our window-specific data (the movie being displayed in the window, the movie controller associated with the movie, and so forth) if we are given the window pointer. On Windows operating systems, we call CreateWindowEx to create a new window and SetWindowLong to store a handle to the window record in the window's data. (See "QuickTime 101" in *MacTech*, January 2000 for a more complete account of all this.)

Now, our entire application framework assumes that each window displayed by our application is associated with a window object record. We need the data in that record to know which movie or image is contained in the window, whether the movie or image has been changed since it was last saved, and similar information. So, when BeginFullScreen returns to us a new fullscreen window, we need to attach a window object to that window. We could, of course, create a new window object and attach it to the window, but things actually work much better if we borrow the window object from the movie window we want to put into fullscreen mode. We'll do that like this:

```
#if TARGET_OS_MAC
     SetWRefCon((**myAppData).fFullScreenWindow,
          (long)theWindowObject);
#endif
#if TARGET_OS_WIN32
     SetWindowLong(GetPortNativeWindow(
          (GrafPtr)(**myAppData).fFullScreenWindow),
          GWL_USERDATA, (LPARAM)theWindowObject);
#endif
```

The Macintosh code is straightforward: simply set the reference constant of the window created by BeginFullScreen to the window object passed to QTBig_StartFullscreen. The Windows code is a little more involved, since BeginFullScreen returns to us a window pointer (of type WindowPtr) but our Windows application expects a movie window to be of type HWND. Accordingly, we need to call GetPortNativeWindow to get the native window (of type HWND) that is associated with the window pointer. This window was created automatically by QuickTime when it created the fullscreen window.

Keep in mind that we now have *two* windows to worry about: (1) the original movie window (which is either of type WindowPtr or HWND, depending on the native operating

system) and (2) a fullscreen window (which is always of type **WindowPtr** but which on Windows is also associated with a window of type **HWND**). We are attaching the window object associated with the original movie window to the fullscreen window; this lets us know which movie to play fullscreen and how to control that movie; it's also important to have a window object available when we want to pass events to the fullscreen movie (as we'll see shortly).

It seems reasonable that once we've created the fullscreen window, we should hide the original movie window; after all, we won't be able to redraw the original movie window or let the user move it around. We can hide the movie window like this:

```
QTFrame_SetWindowVisState(theWindowObject, false);
```

The **QTFrame_SetWindowVisState** function is a new function that we need to add to our framework; it's defined in Listing 4.

Listing 4: Showing or hiding a movie window

QTFrame_SetWindowVisState

```
void QTFrame_SetWindowVisState
        (WindowObject theWindowObject, Boolean theState)
{
  // make sure we have a non-NULL window object and window
  if (theWindowObject == NULL)
    return;

  if ((**theWindowObject).fWindow == NULL)
    return;

  // set the visibility state of the window
#if TARGET_OS_MAC
  if (theState)
    MacShowWindow((**theWindowObject).fWindow);
  else
    HideWindow((**theWindowObject).fWindow);
#endif
#if TARGET_OS_WIN32
  ShowWindow((**theWindowObject).fWindow, theState);
#endif
}
```

(It's interesting to note that QuickTime Player does not hide the original movie window after putting it into fullscreen mode. What's up with that?)

There is one final set of tasks we need to attend to when putting a movie window into fullscreen mode. As we've seen, we are using the same movie and movie controller in the new fullscreen window that we use in the original movie window. But of course the fullscreen window has a different graphics world and a different size. So we need to save the graphics world, size, and a few other pieces of information before we go into fullscreen mode and later restore them when we return to normal mode. **QTBig_StartFullscreen** contains these lines to save the relevant information in our application data record:

```
GetMovieGWorld(myMovie, &(**myAppData).fOrigMovieGWorld,
        NULL);
GetMovieBox(myMovie, &(**myAppData).fOrigMovieRect);
MCGetControllerBoundsRect(myMC,
        &(**myAppData).fOrigControllerRect);
(**myAppData).fOrigControllerVis = MCGetVisible(myMC);
(**myAppData).fOrigControllerAttached =
        MCIsControllerAttached(myMC);
(**myAppData).fOrigWindow = (**theWindowObject).fWindow;
```

Once we've successfully entered fullscreen mode, we need to set the movie graphics world, movie controller bounds, and so forth for the new fullscreen window:

```
SetGWorld(GetWindowPort((**myAppData).fFullScreenWindow),
        NULL);
SetMovieGWorld(myMovie,
GetWindowPort((**myAppData).fFullScreenWindow), NULL);
MCSetControllerPort(myMC,
        GetWindowPort((**myAppData).fFullScreenWindow));
MCSetControllerAttached(myMC, false);
MCSetControllerBoundsRect(myMC, &myRect);

MCSetVisible(myMC, false);
MCActivate(myMC, (**myAppData).fFullScreenWindow, true);
```

We also need to set the **fWindow** field of the window object record to the new fullscreen window:

```
(**theWindowObject).fWindow =
        QTFrame_GetWindowReferenceFromWindow(
        (**myAppData).fFullScreenWindow);
```

And we want to keep track of the fullscreen window in a global variable:

```
gFullScreenWindowObject = theWindowObject;
```

We are finally ready to take a look at the complete definition of **QTBig_StartFullscreen**, shown in Listing 5.

Listing 5: Entering fullscreen mode

QTBig_StartFullscreen

```
OSErr QTBig_StartFullscreen (WindowObject theWindowObject)
{
  MovieController          myMC = NULL;
  Movie                    myMovie = NULL;
  ApplicationDataHdl       myAppData = NULL;
  long                     myFlags = fullScreenAllowEvents;
  OSErr                    myErr = paramErr;

  if (theWindowObject == NULL)
    goto bail;

  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(theWindowObject);
  myMovie = (**theWindowObject).fMovie;
  myMC = (**theWindowObject).fController;

  if ((myAppData == NULL) || (myMovie == NULL) ||
        (myMC == NULL))
    goto bail;

  if ((**myAppData).fFullScreenWindow == NULL) {
    short           myOrigScreenWidth = 0;
    short           myOrigScreenHeight = 0;
    short           myNewScreenWidth = 0;
    short           myNewScreenHeight = 0;
    short           myScreenWidth = 0;
    short           myScreenHeight = 0;
    short           myMovieWidth = 0;
    short           myMovieHeight = 0;
    Fixed           myScreenRatio;
    Fixed           myMovieRatio;
    Rect            myRect;
    RGBColor        myColor = {0x0000, 0x0000, 0x0000};// black

    // remember some of the current state
    GetMovieGWorld(myMovie, &(**myAppData).fOrigMovieGWorld,
        NULL);
    GetMovieBox(myMovie, &(**myAppData).fOrigMovieRect);
```

SOFTWARE & DOCUMENT PROTECTION

Have a look at this

Dongles are out!

You are right.
Simple Dongles are obsolete.

WIBU-KEY is in!

WIBU-KEY on the other hand offers additional benefits for your customers on every platform Mac – PC – Unix:

Software goes online
Electronic Software Distribution – safely protected by WIBU-KEY

Web Remote Programming
Transferring data into the WIBU-BOX at the customer's site

Web Authentication
Authentication of the user with the help of his WIBU-BOX

Pay-Per-Use
Usage dependent accounting

Mac OS 9 & X
WIBU-KEY supports Mac OS, Windows and heterogeneous networks

www.wibu.com

WIBU SYSTEMS

WIBU-SYSTEMS AG
76137 Karlsruhe, Germany
WIBU-SYSTEMS USA, Inc.
Seattle, WA 98101
email: info@wibu.com

1-800-986-6578
Order Your Test Kit

GRIFFIN TECHNOLOGIES

USA, Canada: Griffin Technologies, LLC
phone: (785) 832-2070 · fax: (785) 832-8787
email: sales@griftech.com · www.griftech.com

Argentina info@safield-group.com, **Australia** simone.eckerle@wibu.com, **Belgium** wibu@impakt.be, **Croatia** aries@aries.hr, **Denmark** ivha@danbit.dk, **Finland** finebyte@finebyte.com, **France** info@neol.fr, **Hungary** info@mrsoft.hu, **Japan** info@suncarla.co.jp, **Jordan, Lebanon** starsoft@cyberia.net.lb, **Korea** dhkimm@hanmir.com, **Netherlands** security@unit4.nl, **Portugal** dubit@dubit.pt, **Spain** sem@sinix.net, **Syria** starsoft@cyberia.net.lb, **United Kingdom** info@codework.com, **USA** sales@griftech.com

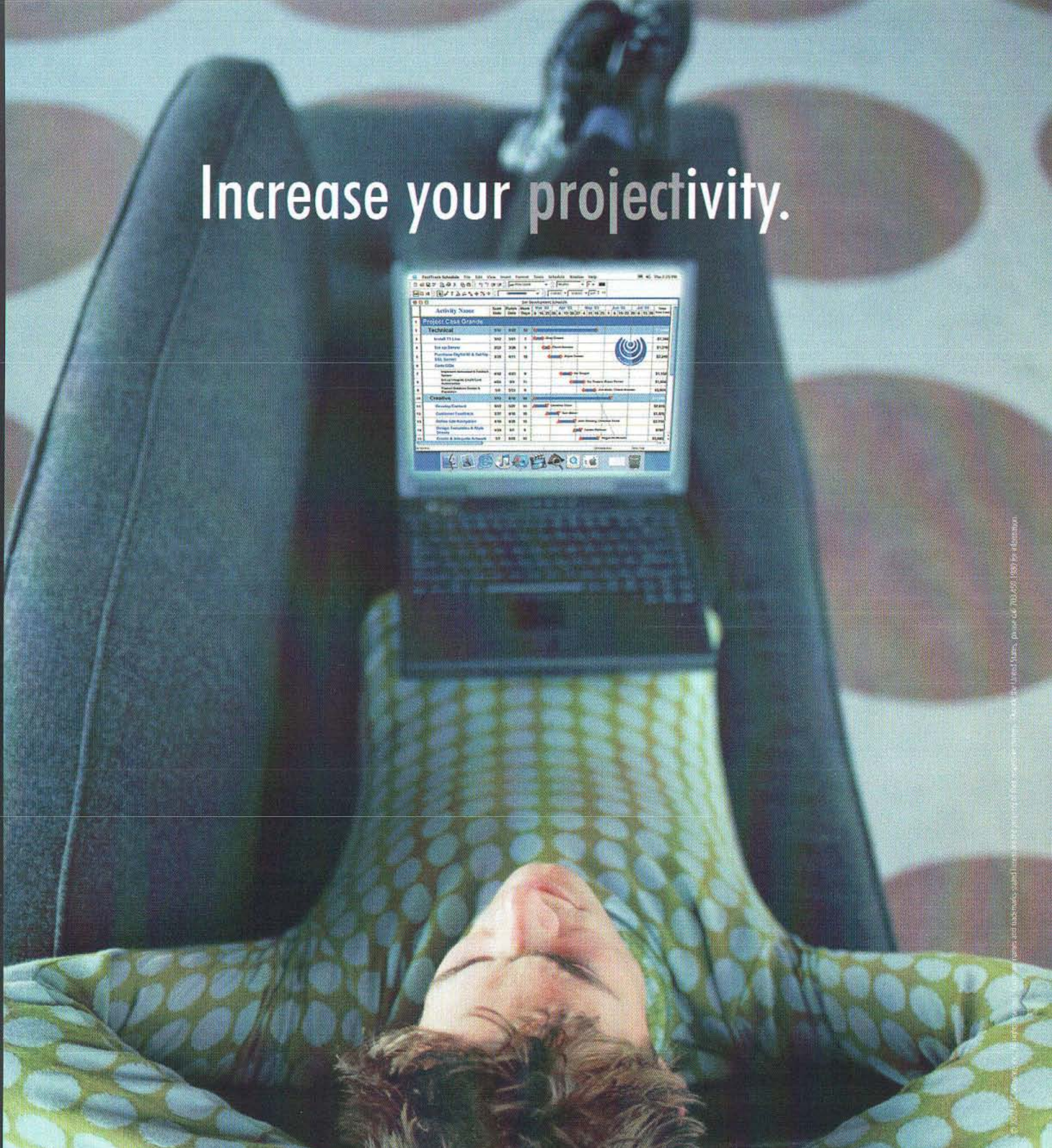
```
     MCGetControllerBoundsRect(myMC,
          &(**myAppData).fOrigControllerRect);
     (**myAppData).fOrigControllerVis = MCGetVisible(myMC);
     (**myAppData).fOrigControllerAttached =
          MCIsControllerAttached(myMC);
     (**myAppData).fOrigWindow = (**theWindowObject).fWindow;

     // get the current screen resolution
     myErr = BeginFullScreen(&(**myAppData).fRestoreState,
          NULL, &myScreenWidth, &myScreenHeight, NULL, NULL,
          fullScreenPreflightSize);
     if (myErr != noErr)
       goto bail;

     // keep track of the original screen resolution
     myOrigScreenHeight = myScreenHeight;
     myOrigScreenWidth = myScreenWidth;

     // calculate the destination rectangle
     GetMovieNaturalBoundsRect(myMovie, &myRect);
     MacOffsetRect(&myRect, -myRect.left, -myRect.top);

     myMovieWidth = myRect.right;
     myMovieHeight = myRect.bottom;

     myMovieRatio = FixRatio(myMovieWidth, myMovieHeight);
     myScreenRatio = FixRatio(myScreenWidth, myScreenHeight);

     // scale the movie rectangle to fit the screen ratio
     if (myMovieRatio > myScreenRatio) {
       myMovieHeight = (myScreenWidth * myMovieHeight) /
                                     myMovieWidth;
       myMovieWidth = myScreenWidth;
     } else {
       myMovieWidth = (myScreenHeight * myMovieWidth) /
                                     myMovieHeight;
       myMovieHeight = myScreenHeight;
     }

     MacSetRect(&myRect, 0, 0, myMovieWidth, myMovieHeight);

     myScreenWidth = myMovieWidth;
     myScreenHeight = myMovieHeight;

     // begin full-screen display
     myErr = BeginFullScreen(&(**myAppData).fRestoreState,
          NULL, &myScreenWidth, &myScreenHeight,
          &(**myAppData).fFullScreenWindow, &myColor,
myFlags);
     if (myErr != noErr)
       goto bail;

     // determine whether the resolution changed; if it has changed, we must have
     // passed in a supported resolution, so we want the movie to fill the screen;
     // otherwise, we need to offset the movie to center it in the screen
     if ((myScreenWidth == myOrigScreenWidth) &&
            (myScreenHeight == myOrigScreenHeight))
       MacOffsetRect(&myRect,
            (myScreenWidth - myMovieWidth) / 2,
            (myScreenHeight - myMovieHeight) / 2);

#if TARGET_OS_WIN32
     // on Windows, set a window procedure for the new window
     QTMLSetWindowWndProc((**myAppData).fFullScreenWindow,
          QTBig_HandleMessages);
#endif

     // hide the original window
     QTFrame_SetWindowVisState(theWindowObject, false);

     // attach the existing window object to the new window
#if TARGET_OS_MAC
     SetWRefCon((**myAppData).fFullScreenWindow,
          (long)theWindowObject);
#endif
#if TARGET_OS_WIN32
     SetWindowLong(GetPortNativeWindow(
          (GrafPtr)(**myAppData).fFullScreenWindow),
          GWL_USERDATA, (LPARAM)theWindowObject);
#endif

     // set the movie and movie controller state to the new window and rectangle
     SetGWorld(GetWindowPort((**myAppData).fFullScreenWindow),
          NULL);
     SetMovieGWorld(myMovie,
          GetWindowPort((**myAppData).fFullScreenWindow),
          NULL);
     MCSetControllerPort(myMC,
          GetWindowPort((**myAppData).fFullScreenWindow));
     MCSetControllerAttached(myMC, false);
     MCSetControllerBoundsRect(myMC, &myRect);

     SetMovieBox(myMovie, &myRect);
     MCSetVisible(myMC, false);
     MCActivate(myMC, (**myAppData).fFullScreenWindow, true);

     (**theWindowObject).fWindow =
          QTFrame_GetWindowReferenceFromWindow(
          (**myAppData).fFullScreenWindow);

#if TARGET_API_MAC_CARBON
     HiliteWindow((**myAppData).fFullScreenWindow, true);
#endif

#if END_FULLSCREEN_AT_MOVIE_END
     // install a callback procedure to return linear, non-looping movies
     // to normal mode at the end of the movie
     if (QTBig_MovieIsStoppable(myMC))
       QTBig_InstallCallBack(theWindowObject);
#endif
   }

   gFullScreenWindowObject = theWindowObject;

bail:
   return(myErr);
}
```

You'll notice a few lines of code we haven't discussed yet. The Windows-only call to QTMLSetWindowWndProc is necessary to handle messages targeted at the fullscreen window, as we'll see in the next section. The Carbon-only call to HiliteWindow ensures that the window returned by BeginFullScreen is highlighted. (Apparently, in some circumstances, that window is occasionally unhighlighted, which blocks any interactivity in the movie.) And the code selected by the END_FULLSCREEN_AT_MOVIE_END flag installs a time base callback function that returns a non-interactive, non-looping movie to normal mode when the movie reaches its end. We'll delve into that issue toward the end of this article.

Handling Events for the Fullscreen Window

Now we've created a fullscreen window, attached a movie and movie controller to it, and positioned the movie box so that the movie appears centered in the fullscreen window. We've also set the movie and movie controller graphics ports to the fullscreen window and hidden the original movie window. So things are looking great. We'd also like them to start *acting* great. That is, we want the fullscreen movie to respond in the normal ways to mouse movements, mouse clicks, and key presses, and we want it to jump out of fullscreen mode if the user presses the Escape key.

This Escape key behavior is quite easy to implement. Our application framework contains a stub function QTApp_HandleKeyPress that we can use to intercept key presses. In QTBigScreen, QTApp_HandleKeyPress is defined as shown in Listing 6. As you can see, we look for presses



Be productive. Stay productive.

Introducing FastTrack Schedule 8. Redesigned for Mac OS X and packed with new productivity features and a bold Aqua interface–FastTrack Schedule 8 has all the tools you need to ensure project success. For a free demo version or to order, call us today at 800.450.1983.

www.fasttrackschedule8.com

Built for Mac OS X

AEC SOFTWARE

on the Escape key while a fullscreen window is active; if we find one, we call QTBig_StopFullscreen.

Listing 6: Handling key presses

QTApp_HandleKeyPress

```
Boolean QTApp_HandleKeyPress (char theCharCode)
{
  Boolean            isHandled = false;

  switch (theCharCode) {
    // Escape key during fullscreen display means to restore the window to its
original
    // state
    case kEscapeCharCode:
      if (gFullScreenWindowObject != NULL)
        isHandled = (QTBig_StopFullscreen
                       (gFullScreenWindowObject) == noErr);
      break;
  }

  return(isHandled);
}
```

All the other important behaviors we care about are provided by the movie controller. So we need to make sure that we call MCIsPlayerEvent for any events our application receives while there is a fullscreen movie. Again we'll insert some code into another application function, QTApp_HandleEvent:

```
if ((**myAppData).fFullScreenWindow != NULL)
  return(MCIsPlayerEvent((**myWindowObject).fController,
         theEvent));
```

On the Macintosh, this is pretty much all we need to do to make the fullscreen movie behave as expected. That's because our Mac framework calls QTApp_HandleEvent for every event it receives, and it calls QTApp_HandleKeyPress on every key event. On Windows, things are a bit trickier. Our Windows framework calls these functions only for MDI child movies — that is, windows using the QTFrame_MovieWndProc window procedure. But the fullscreen window is not an MDI child window. The easy solution here is to define a custom window procedure for the fullscreen window that calls QTApp_HandleEvent and QTApp_HandleKeyPress at the appropriate times. (We install this window procedure by calling QTMLSetWindowWndProc, as you saw in Listing 5.) Listing 7 shows our definition of QTBig_HandleMessages.

Listing 7: Handling fullscreen window messages

QTBig_HandleMessages

```
LRESULT CALLBACK QTBig_HandleMessages (HWND theWnd,
        UINT theMessage, UINT wParam, LONG lParam)
{
  MovieController      myMC = NULL;
  Movie                myMovie = NULL;
  WindowObject         myWindowObject = NULL;
  MSG                  myMsg = {0};
  EventRecord        myMacEvent;
  LONG                 myPoints = GetMessagePos();
  BOOL                 myIsHandled = false;

  if (gFullScreenWindowObject == NULL)
    goto bail;

  // make sure we don't get called while the movie is returning to normal state
  if (gEndingFullScreen)
    goto bail;

  // get the window object, movie, and movie controller for this window
  myWindowObject = gFullScreenWindowObject;
  myMC = (**myWindowObject).fController;
  myMovie = (**myWindowObject).fMovie;

  // give the movie controller this message first
  if (myMC != NULL) {
    LONG                myPoints = GetMessagePos();

    myMsg.hwnd = theWnd;
    myMsg.message = theMessage;
    myMsg.wParam = wParam;
    myMsg.lParam = lParam;
    myMsg.time = GetMessageTime();
    myMsg.pt.x = LOWORD(myPoints);
    myMsg.pt.y = HIWORD(myPoints);

    // translate a Windows event to a Mac event
    WinEventToMacEvent(&myMsg, &myMacEvent);

    // let the application-specific code have a chance to intercept the event
    myIsHandled = QTApp_HandleEvent(&myMacEvent);
  }

  switch (theMessage) {
    case WM_CHAR:
      // do any application-specific key press handling
      myIsHandled = QTApp_HandleKeyPress((char)wParam);
      break;
  }

bail:
  return(DefWindowProc(theWnd, theMessage, wParam, lParam));
}
```

There's nothing too extravagant here. Notice, however, that we look at the gEndingFullScreen global variable to see whether we got this message during a call to EndFullScreen. If so, we don't want to pass the event to the movie controller.

We need to make a couple of other small changes to get everything working satisfactorily, now that our application has to support fullscreen windows. For instance, in the function QTFrame_SizeWindowToMovie, we use code like this to get the size of a movie that is associated with a movie controller:

```
if (myMC != NULL)
  if (MCGetVisible(myMC))
    MCGetControllerBoundsRect(myMC, &myRect);
```

This assumes that the controller is attached to the movie, which is always true for our movie windows but false for our fullscreen movie. So we need to revise that code like so:

```
if (myMC != NULL)
  if (MCGetVisible(myMC))
    if (MCIsControllerAttached(myMC)
      MCGetControllerBoundsRect(myMC, &myRect);
```

Exiting Fullscreen Mode

Returning from fullscreen mode to normal windowed mode is a breeze. The main step is of course to call EndFullScreen. To prevent QTBig_HandleMessages from being called while EndFullScreen is executing, we bracket our call with setting and unsetting the gEndingFullScreen global variable:

```
gEndingFullScreen = true;
myErr = EndFullScreen((**myAppData).fRestoreState, 0L);
```

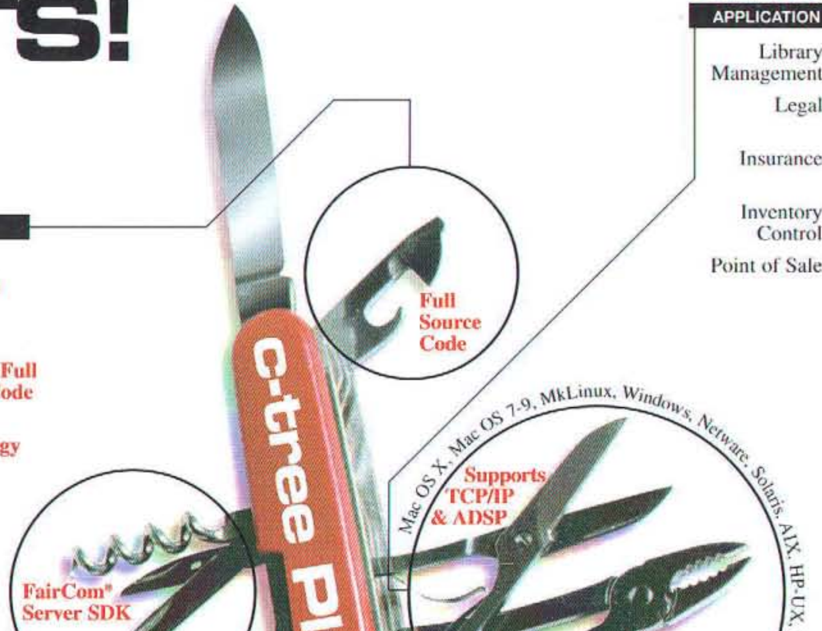
CUT YOUR DATABASE COSTS!

Embedded Hardware

APPLICATION	BENEFIT
Internet Appliances	**Small footprint**
Medical Devices	**Stable**
Factory Automation	**Includes Full Source Code**
Space Exploration	**Proven Technology**

Vertical Markets

APPLICATION	BENEFIT
Library Management	**Easy Deployment**
Legal	**Flexible Licensing**
Insurance	**Huge File Support**
Inventory Control	**Scalable**
Point of Sale	**Cross Platform**


c-tree Plus®
Full Source Code
Mac OS X, Mac OS 7-9, MkLinux, Windows, Netware, Solaris, AIX, HP-UX, Linux, SCO, QNX, FreeBSD, and more.
Supports TCP/IP & ADSP
FairCom® Server SDK
Cross Platform

Corporate-Wide Applications

APPLICATION	BENEFIT
B2B Server	**Flexible Communication**
Proprietary Applications	**Fast**
Dedicated Web Server	**Robust Threading**

C-TREE DELIVERS FAST, SOLID, AND FLEXIBLE DATABASE TECHNOLOGY WHILE CUTTING YOUR DEPLOYMENT COSTS...

For over twenty years, FairCom's proven database technology has been a favorite of commercial developers for Macintosh applications ranging from low level embedded appliances to vertical market applications to huge multi-platform, corporate applications. FairCom has established this reputation by delivering uncompromising Standalone and Client/Server technology in a flexible package that enables you to cut costs while utilizing cutting edge technology like our new HUGE file support and 64-bit support. FairCom offers this exceptional performance, unsurpassed data availability and rock solid reliability with a low total cost of ownership and flexible licensing options.

Get more for your development dollar! **Visit www.faircom.com/ep/freecdoffer** today for a free developer's CD.

FairCom® CORPORATION

FairCom	Offices
USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

Mac Support Since 1985 • www.faircom.com • USA. 800.234.8180 • info@faircom.com

```
gEndingFullScreen = false;
```

We also need to restore the settings we saved previously, such as the movie and movie controller graphics ports and the visibility state of the controller bar. It's all pretty much what you'd expect (Listing 8).

Listing 8: Returning to normal window mode

QTBig_StopFullscreen

```
OSErr QTBig_StopFullscreen (WindowObject theWindowObject)
{
  ApplicationDataHdl      myAppData = NULL;
  OSErr                   myErr = paramErr;

  if (theWindowObject == NULL)
    goto bail;

  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(theWindowObject);
  if (myAppData != NULL) {
    if ((**myAppData).fFullScreenWindow != NULL) {

      // restore the original settings
      (**theWindowObject).fWindow =
                  (**myAppData).fOrigWindow;

      MacSetPort(QTFrame_GetPortFromWindowReference(
        (**theWindowObject).fWindow));

      SetMovieGWorld((**theWindowObject).fMovie,
        (CGrafPtr)(**myAppData).fOrigMovieGWorld,
        GetGWorldDevice(
            (CGrafPtr)(**myAppData).fOrigMovieGWorld));
      SetMovieBox((**theWindowObject).fMovie,
        &(**myAppData).fOrigMovieRect);

      MCSetControllerPort((**theWindowObject).fController,
        (CGrafPtr)(**myAppData).fOrigMovieGWorld);
      MCSetControllerAttached(
        (**theWindowObject).fController,
        (**myAppData).fOrigControllerAttached);
      MCSetVisible((**theWindowObject).fController,
        (**myAppData).fOrigControllerVis);
      MCSetControllerBoundsRect(
        (**theWindowObject).fController,
        &(**myAppData).fOrigControllerRect);

      gEndingFullScreen = true;

      // end fullscreen playback
      myErr = EndFullScreen((**myAppData).fRestoreState,
0L);

      gEndingFullScreen = false;

      // empty out the data structures and global variables
      (**myAppData).fOrigWindow = NULL;
      (**myAppData).fFullScreenWindow = NULL;
      (**myAppData).fRestoreState = NULL;
      (**myAppData).fOrigMovieGWorld = NULL;

      gFullScreenWindowObject = NULL;

#if END_FULLSCREEN_AT_MOVIE_END
      // dispose of the CallMeWhen callback and the callback UPP
      if ((**myAppData).fCallBack != NULL)
        DisposeCallBack((**myAppData).fCallBack);

      if ((**myAppData).fCallBackUPP != NULL)
        DisposeQTCallBackUPP((**myAppData).fCallBackUPP);

      (**myAppData).fCallBack = NULL;
      (**myAppData).fCallBackUPP = NULL;
#endif

      // make sure the movie window is the correct size and then show it again
      QTFrame_SizeWindowToMovie(theWindowObject);
      QTFrame_SetWindowVisState(theWindowObject, true);
    }
  }

bail:
  return(myErr);
}
```

For now, don't worry about the code in the END_FULLSCREEN_AT_MOVIE_END block; we'll discuss that a bit later.

Flash Application Messages

In a recent article ("The Flash", cited earlier), we learned that Flash movies can send messages to the playback application requesting that it perform specific actions such as quitting or launching some other application. Of the five standard *Flash application messages*, or FSCommands, one is relevant to our current concern: fullscreen. This command can have one of two arguments, either true or false. If the argument is true, then the playback application should put the window containing the Flash movie into fullscreen mode; if it's false, then the application should put the window into normal mode. We are now able to upgrade the sample application we built in that article, QTFlash, to support the fullscreen application message.

QuickTime's Flash media handler intercepts Flash application messages and repackages them into movie controller actions of type mcActionDoScript, which are then sent to the application's movie controller action filter function. In QTFlash, we responded to those actions by calling the application function QTFlash_DoFSCommand, like so:

```
case mcActionDoScript:
  isHandled = QTFlash_DoFSCommand(theMC,
        (QTDoScriptPtr)theParams, myWindowObject);
  break;
```

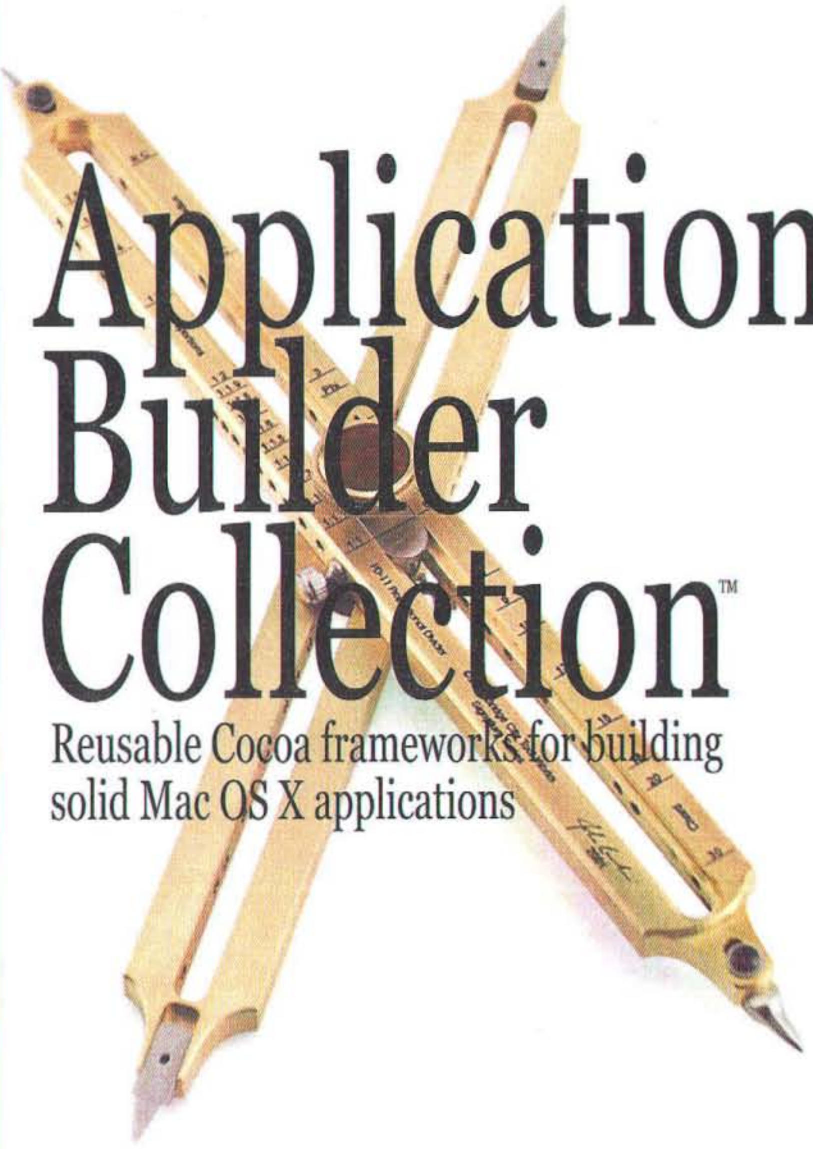
QTFlash_DoFSCommand, in turn, inspected the command and argument fields of the specified QTDoScriptRecord to determine which command to perform. We can now add the lines in Listing 9 to support entering and exiting fullscreen mode:

Listing 9: Handling the fullscreen application message

QTFlash_DoFSCommand

```
if (strcmp(theScriptPtr->command, "fullscreen") == 0) {
  if (strcmp(theScriptPtr->arguments, "true") == 0)
    QTBig_StartFullscreen(theWindowObject);

  if (strcmp(theScriptPtr->arguments, "false") == 0)
    QTBig_StopFullscreen(theWindowObject);

  isHandled = true;
}
```

As you can see, we call our new functions QTBig_StartFullscreen and QTBig_StopFullscreen. Of course we also need to add to QTFlash all the additional code in QTBigScreen that handles fullscreen mode (such as the code in QTApp_HandleEvent).



Application Builder Collection™

Reusable Cocoa frameworks for building solid Mac OS X applications

- Port to Cocoa faster
- Save thousands by reducing coding and testing time
- Spend your time developing the features that make your software stand out
- Quickly create professional looking applications
- Utilize object-oriented technology for easy customization and reuse
- No deployment licenses or royalties!

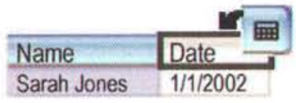
Collection Includes:

- Rotational Slider
- Formatters for Currency, Phone, Text, and Custom Fields
- Draggable Cocoa Cells for Tables
- Calendar View and Popup
- URL Clickable Field
- Twiddle View
- Archive Builder Application
- And More . . .

Learn more at www.jiiva.com

rotate images and more

control data input and presentation



drag sliders and other widgets into tables

add calendar popups and views

turn text fields into clickable URLs

create complex interfaces with versatile twiddles

Mac

Built for Mac OS X

Jiiva, Inc.
www.jiiva.com/abc

QuickTime Application Messages

Interestingly, QuickTime 5.0.1 introduced a mechanism for sending messages from a QuickTime movie to the playback application that is strongly reminiscent of the FSCommand capability in Flash movies. These *QuickTime application messages* provide a way for a movie to request fullscreen mode for the window containing the movie, close the window containing the movie, and perform several other tasks.

The standard way to send an application message from a QuickTime movie to the playback application is by using the kActionSendAppMessage wired action. This action requires one parameter, which is of type long and which specifies the message to send to the application. Currently QuickTime defines five public application messages (in the file Movies.h):

```
enum {
  kQTAppMessageSoftwareChanged                        = 1,
  kQTAppMessageWindowCloseRequested           = 3,
  kQTAppMessageExitFullScreenRequested        = 4,
  kQTAppMessageDisplayChannels                        = 5,
  kQTAppMessageEnterFullScreenRequested       = 6
};
```

The kQTAppMessageSoftwareChanged message indicates that some part of the QuickTime software has been updated, and the kQTAppMessageDisplayChannels message requests that the QuickTime Player application display its user interface for selecting one of the QuickTime channels; neither of these messages is relevant to our sample applications and we shall blithely ignore them. The kQTAppMessageEnterFullScreenRequested and kQTAppMessageExitFullScreenRequested messages request that the playback application put the associated movie window into fullscreen or normal mode, and the kQTAppMessageWindowCloseRequested message requests that the playback application close the window containing the movie.

We insert one of these application messages into a movie by inserting a wired action in the standard manner. For instance, Listing 10 shows how we can make a mouse click on a sprite close the movie that contains it.

Listing 10: Adding a window-close request to a sprite

QTWired_AddWindowCloseActionToSprite

```
myErr = QTInsertChild(mySpriteAtom, kParentAtomIsContainer,
         kQTEventType, kQTEventMouseClickEndTriggerButton, 1,
         0, NULL, &myEventAtom);
if (myErr != noErr)
  goto bail;

// add an action atom to the event handler
myErr = QTInsertChild(mySpriteAtom, myEventAtom, kAction, 0,
         0, 0, NULL, &myActionAtom);
if (myErr != noErr)
  goto bail;

myAction = EndianU32_NtoB(kActionSendAppMessage);
myErr = QTInsertChild(mySpriteAtom, myActionAtom,
         kWhichAction, 1, 1, sizeof(myAction), &myAction,
         NULL);
if (myErr != noErr)
  goto bail;

// add a parameter atom to specify which action to perform
myMsg = EndianU32_NtoB(kQTAppMessageWindowCloseRequested);
myErr = QTInsertChild(mySpriteAtom, myActionAtom,
         kActionParameter, 0, 1, sizeof(myMsg), &myMsg,
         NULL);
```

We add an event atom whose atom ID is kQTEventMouseClickEndTriggerButton to the sprite atom (mySpriteAtom). Then we insert an action atom into that event atom. The action atom is given two children, one of type kWhichAction whose atom data is kActionSendAppMessage, and one of type kActionParameter whose atom data is kQTAppMessageWindowCloseRequested.

Keep in mind that a movie controller cannot handle these application messages by itself. The actions requested here (entering or exiting fullscreen mode, and closing the window containing a movie) require assistance from the playback application. Accordingly, the movie controller informs the application of the request by sending it a movie controller action of type mcActionAppMessageReceived. The application is free to act on that request or ignore it entirely. In this section, we'll see how to handle the fullscreen and close-window application messages.

Handling Fullscreen Messages

It's fairly simple to add support for the kQTAppMessageEnterFullScreenRequested and kQTAppMessageExitFullScreenRequested application messages to QTBigScreen. Listing 11 shows the code we'll add to our movie controller action filter procedure to handle these messages.

Listing 11: Handling application messages

QTApp_MCActionFilterProc

```
case mcActionAppMessageReceived:
  switch ((long)theParams) {
    case kQTAppMessageEnterFullScreenRequested:
      QTBig_StartFullscreen(myWindowObject);
      isHandled = true;
      break;

    case kQTAppMessageExitFullScreenRequested:
      QTBig_StopFullscreen(myWindowObject);
      isHandled = true;
      break;
  }

  break;
```

Handling Close-Window Messages

Now let's see how to handle the kQTAppMessageWindowCloseRequested application message. It's tempting perhaps to handle this message by adding these few lines of code to the switch statement in Listing 11:

```
    case kQTAppMessageWindowCloseRequested:
      QTFrame_DestroyMovieWindow((**myWindowObject).fWindow);
      isHandled = true;
      break;
```

Here, we simply call the framework function QTFrame_DestroyMovieWindow to close the window associated with the specified movie. This function checks to see whether the movie has been changed since it was opened or last saved and, if so, prompts the user to save or discard any changes; once the changes have been saved or discarded, QTFrame_DestroyMovieWindow calls QTFrame_CloseWindowObject to close the movie file and dispose of the movie and movie controller. Then QTFrame_DestroyMovieWindow calls DisposeWindow (on Macintosh systems) or SendMessage with the WM_MDIDESTROY message (on Windows systems) to actually close and destroy the movie window.

There is a problem with this simple approach, however. As noted, QTFrame_CloseWindowObject disposes of the movie controller associated with the movie, and experience tells me that this is definitely a bad thing to do inside of a movie controller action filter procedure. So we need to adopt a different approach: in response to the kQTAppMessageWindowCloseRequested message, we'll set a flag in our application data record that indicates that we need to destroy the window object. Listing 12 shows the code we'll add.

Listing 12: Handling close-window messages

QTApp_MCActionFilterProc

```
case kQTAppMessageWindowCloseRequested:
  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(myWindowObject);
  if (myAppData != NULL)
    (**myAppData).fDestroyWindowNeeded = true;
  isHandled = true;
  break;
```

Later, in the QTApp_HandleEvent function, we'll need to cycle through all open movie windows to see whether any of them needs to be closed. We'll add a while loop, as shown in Listing 13.

Listing 13: Looking for windows to close

QTApp_HandleEvent

```
myWindow = QTFrame_GetFrontMovieWindow();
while (myWindow != NULL) {
  myNextWindow = QTFrame_GetNextMovieWindow(myWindow);

  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindow(myWindow);
  if (myAppData != NULL) {
    if ((**myAppData).fDestroyWindowNeeded) {
      (**myAppData).fDestroyWindowNeeded = false;
      QTFrame_DestroyMovieWindow(myWindow);
    }
  }

  myWindow = myNextWindow;
}
```

PRESENTATION MOVIE USER DATA

A QuickTime movie file can contain a piece of movie user data of type 'ptv ' that specifies that the movie should be *presented* — that is, displayed at a certain size against a solid black background. When a movie is presented, it's drawn without the normal window frame or controller bar. Most often, this feature is used to put a movie into fullscreen mode automatically when the movie file is opened. QuickTime Player

THE #1 RELATIONAL DATABASE ON MAC OS X

OPENBASE SQL 7.0

HOW IMPORTANT IS YOUR DATA?

With its redundant, multi-file design,

OpenBase SQL

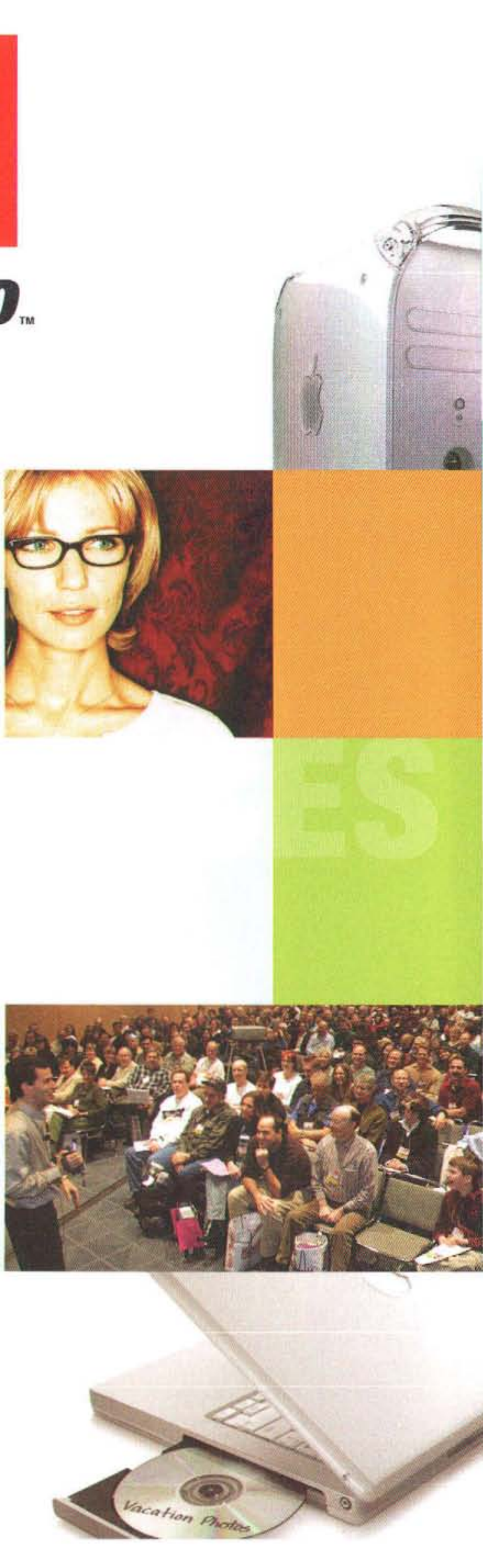
ensures data integrity and availability,

while enhancing database performance.

OPENBASE® INTERNATIONAL

For more info, visit:
www.openbase.com

Macworld
Conference & Expo™



Register online **www.macworldexpo.com**

Conferences **July 15-19, 2002**
Expo **July 17-19, 2002**

Jacob K. Javits Center **New York**

For more information, call toll free
1-800-645-EXPO

IDG WORLD EXPO

See hundreds of companies and thousands of products at the largest technology show in New York.

Macworld Conference & Expo is more than a conference, and more than an exposition. It's a MUST ATTEND staple for the Mac community. Enhance your knowledge, network with peers, personally interact with new products and technologies, and finalize your purchase decisions under one roof.

Personalize your educational experience, by mixing-and-matching conference programs.

- **Macworld/Users** – Learn tips and tricks for your favorite application, how to maximize you digital capabilities, get a taste of Mac OS X.
- **Macworld/Power Tools** – Immerse yourself for two days with one of the most popular productivity tools for the Mac, and take your skill set to its top-level.
- **Macworld/Pro** – Participate in sophisticated training for Mac networking, digital video and filmmaking, professional publishing, Mac systems administrations and management, and detailed technical presentations that take you inside Mac OS X.
- **MacBeginnings** – Enjoy educational sessions full of tips, techniques and fact-filled training. Learn Mac basics, or about the Internet. Learn how to set-up and create desktop movies, or how to join and utilize a Macintosh user group. These sessions are open to all registered attendees.
- **Workshops** – Make the most out of your show experience by adding a full-day workshop to your educational agenda.
- **Brand New, Hands-on MacLabs** – Provide hands-on computer training on key applications and tools. Our trainers are experts in their field, and they are prepared to share their knowledge with you so select a discipline to focus on and bring your laptop!

Register online today using Priority Code: A-MTA
www.macworldexpo.com

Flagship Sponsors

Macworld Macworld.com MacCentral

looks for and interprets this user data item, and in this section we'll see how to handle it in QTBigScreen (at least in part).

The data contained in a 'ptv ' user data item occupies 8 bytes; we can exhibit the structure of this data using a QTPFSDataRec structure, defined like this:

```
typedef struct {
  UInt16        fSize;
  UInt16        fUnused1;
  UInt16        fUnused2;
  Boolean       fPlaySlideShow;
  Boolean       fPlayOnOpen;
} QTPFSDataRec;
```

This structure is defined in our file QTBigScreen.h; there is no 'ptv ' item structure defined in any of the public QuickTime header files. (By the way, "ptv" stands for "print to video"; this is because the presenting capability was intended also to provide a non-windowed version of a movie that could be written out to videotape.)

The fSize field contains a 16-bit integer that specifies the desired size of the presented movie. The value 0 indicates that the movie should be presented at its normal size. The value 1 indicates that the movie should be presented at twice its normal size. The value 2 indicates that the movie should be presented at half its normal size. The value 3 indicates that the movie should be presented fullscreen. QuickTime Player calls BeginFullScreen with the appropriate width and height to present the movie. As we've seen, however, this can cause the resolution of the screen to be changed, in which case the actual size of the presented movie might not be what we would expect. To force QuickTime Player to present the movie at its normal size, we can specify the value 4. We'll define these constants for the fSize field:

```
enum {
  kSizeNormal              = 0,
  kSizeDouble              = 1,
  kSizeHalf                = 2,
  kSizeFullScreen             = 3,
  kSizeCurrent             = 4
};
```

The values of these constants appear to derive from the order of the items in the Movie Size pop-up menu in the dialog box displayed by QuickTime Player when the user selects the "Present Movie..." item in the Movie menu (shown in Figure 3).

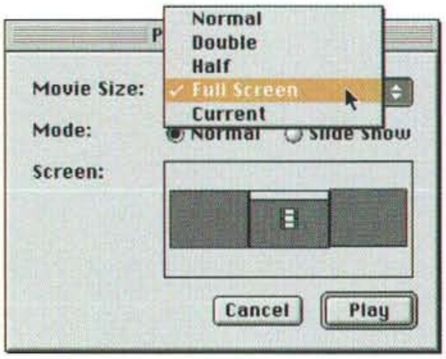


***Figure 3**: QuickTime Player's Present Movie dialog box*

The fPlaySlideShow field contains a Boolean value that indicates whether the movie should be played in *slide-show mode*; in slide-show mode, the movie advances to another frame only when the user presses the right-arrow or left-arrow key (which move the movie forward one frame or backward one frame, respectively). Any sounds tracks in the movie are ignored during slide-show mode. Finally, the fPlayOnOpen field contains a Boolean value that indicates whether the movie should begin playing automatically when the movie is presented.

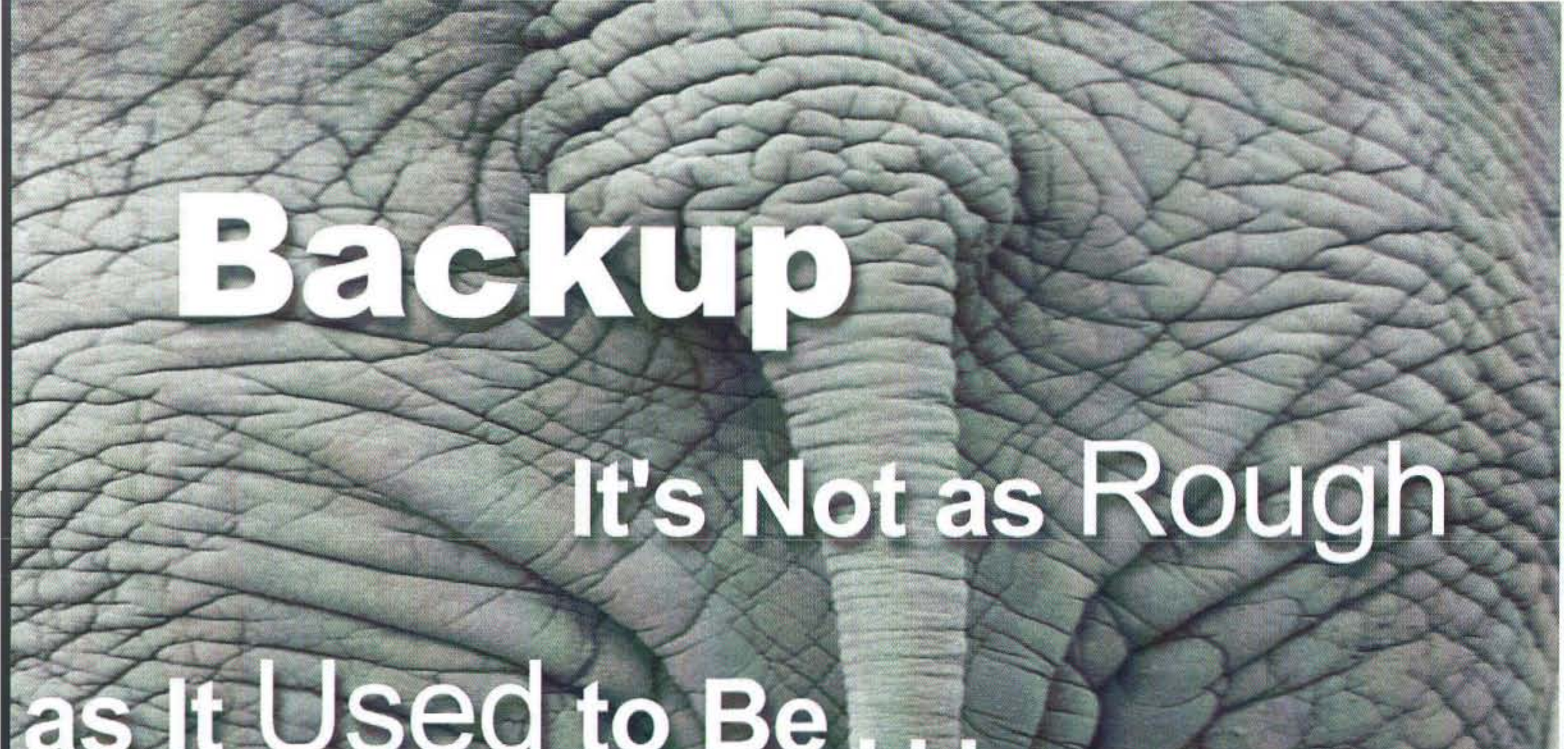
Listing 14 defines a function that we can use to add a 'ptv ' user data item to a movie.

Listing 14: Adding a play-fullscreen item to a movie

QTBig_AddPTVItemToMovie

```
#define kPTVItemType          FOUR_CHAR_CODE('ptv ')

OSErr QTBig_AddPTVItemToMovie (Movie theMovie)
{
  UserData          myUserData = NULL;
  short             myCount = 0;
  QTPFSDataRec      myPFSData;
  OSErr             myErr = noErr;

  // get the movie's user data list
  myUserData = GetMovieUserData(theMovie);
  if (myUserData == NULL)
    return(paramErr);

  // we want to end up with at most one user data item of type 'ptv ',
  // so let's remove any existing ones
  myCount = CountUserDataType(myUserData, kPTVItemType);
  while (myCount--)
    RemoveUserData(myUserData, kPTVItemType, 1);

  // add a new user data item of type 'ptv '
  myPFSData.fSize = EndianU16_NtoB(kSizeFullScreen);
  myPFSData.fUnused1 = 0;
  myPFSData.fUnused2 = 0;
  myPFSData.fPlaySlideShow = false;
  myPFSData.fPlayOnOpen = true;
  myErr = SetUserDataItem(myUserData, &myPFSData,
        sizeof(myPFSData), kPTVItemType, 0);

  return(myErr);
}
```

QTBig_AddPTVItemToMovie adds to the specified movie a 'ptv ' user data item that requests that the movie be played back



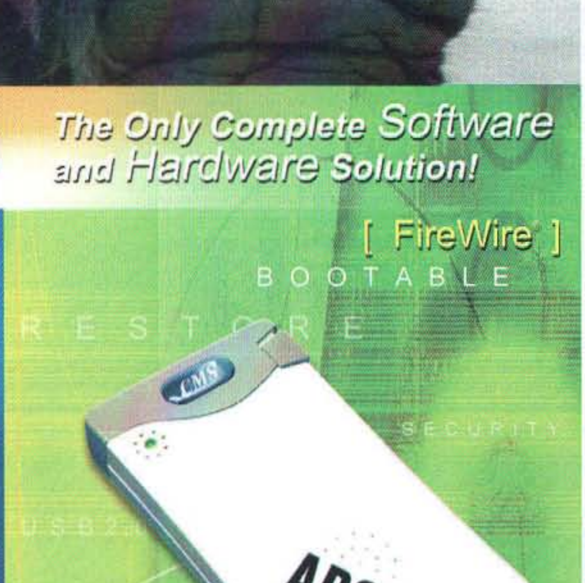
utomatic Backup System

The Only Complete Software and Hardware Solution!

he ABS is The Easiest Way Invented
Backup Your Mac®... Automatically!

ackup, store & share files between iMacs™, iBooks™,
ower Macs™, & PowerBooks™, including video,
gital photography, graphics, & MP3s up to **60GB**!

r More Information Call **800 • 327 • 5773** or
lick www.**cmsproducts.com**/mt1



CMS PERIPHERALS

Now Get 15% Off
All Online Orders!

Mac

fullscreen and that the movie start playing as soon as it's opened. (It's worth mentioning that a movie can also contain a user data item of type 'ptvc', whose data is an RGBColor structure that specifies the background color of a presented movie.)

Now, when QTBigScreen opens a movie file that contains a movie user data item of type 'ptv ' and the value in the fSize field is kSizeFullScreen, we want to call our function QTBig_StartFullscreen to play it fullscreen. We also want to inspect the fPlayOnOpen field to see whether the movie should begin playing immediately. We'll add the lines of code in Listing 15 to our framework function QTFrame_OpenMovieInWindow.

Listing 15: Processing a 'ptv ' user data item

QTFrame_OpenMovieInWindow

```
UserData        myUserData = NULL;
QTPFSDataRec    myRec;
OSErr           myErr = paramErr;

// get the movie's user data list
myUserData = GetMovieUserData(myMovie);
if (myUserData != NULL) {
  myErr = GetUserDataItem(myUserData, &myRec, sizeof(myRec),
        FOUR_CHAR_CODE('ptv '), 0);
  if (myErr == noErr) {
    myRec.fSize = EndianU16_BtoN(myRec.fSize);

    if (myRec.fSize == kSizeFullScreen)
      MCDoAction(myMC, mcActionAppMessageReceived,
            (void *)kQTAppMessageEnterFullScreenRequested);

    if (myRec.fPlayOnOpen)
      MCDoAction(myMC, mcActionPrerollAndPlay,
            (void *)GetMoviePreferredRate(myMovie));
  }
}
```

Notice that we don't call QTBig_StartFullscreen directly; instead, we issue a movie controller action of type mcActionAppMessageReceived with the parameter kQTAppMessageEnterFullScreenRequested, which (as we've seen) causes QTBigScreen to call QTBig_StartFullscreen. Notice also that we ignore all size values except kSizeFullScreen; I'll leave it as an exercise for the motivated reader to modify QTBigScreen so that it can present a movie in any of the currently defined sizes.

Time Base Callback Functions

We noted earlier that QuickTime Player automatically returns from fullscreen mode to normal mode when it reaches the end of a non-interactive, non-looped movie. We can achieve this same behavior in QTBigScreen by installing a *time base callback function*, a function that is executed when a specific time in a movie is reached or when some other event related to the movie's time base occurs.

You may recall from previous articles that a movie's time base controls the direction and speed of movie playback, as well as its looping state and current movie time. A time base (of type TimeBase) is automatically created when we load a movie; we can retrieve a movie's time base at any time by calling the GetMovieTimeBase function. We can attach to a time base one or more callback functions that are triggered when a specific *callback event* occurs. Currently, five types of callback events are defined; we indicate a specific type of callback event using these constants:

```
enum {
  callBackAtTime                = 1,
  callBackAtRate                = 2,
  callBackAtTimeJump            = 3,
  callBackAtExtremes            = 4,
  callBackAtTimeBaseDisposed    = 5
};
```

The callBackAtTime event causes a callback function to be called when a specific time value in the movie is reached (for instance, when the movie reaches the 2 second mark). The callBackAtRate event causes a callback function to be called when the movie begins to play at a specified rate (for instance, when the movie is played at twice the normal speed). The callBackAtTimeJump event causes a callback function to be called when a jump in time occurs; this means that the callback function is called whenever the movie time is set to a time different from what it would be set to under normal movie playback (for instance, if the user clicks in the movie controller bar to select a different movie time, or if a wired action changes the current movie time). The callBackAtTimeBaseDisposed event causes a callback function to be called when the time base is about to be disposed.

For present purposes, we are interested in the callBackAtExtremes event, which occurs at either the beginning or the end of the movie. As we'll see in a moment, we indicate that we want our callback function to be called at the end of the movie by passing a specific value when we activate the time base callback.

In QTBigScreen, we call the QTBig_InstallCallBack function to install a time base callback function to return from fullscreen mode to normal mode at the end of the movie. We add these lines of code to QTBig_StartFullscreen:

```
#if END_FULLSCREEN_AT_MOVIE_END
    // install a callback procedure to return linear, non-looping movies
    // to normal mode at the end of the movie
    if (QTBig_MovieIsStoppable(myMC))
      QTBig_InstallCallBack(theWindowObject);
#endif
```

The definition of QTBig_MovieIsStoppable is quite simple; we just call MCGetControllerInfo to determine whether the movie is looping or interactive, as shown in Listing 16.

Listing 16: Determining whether a movie should be stopped

QTBig_MovieIsStoppable

```
Boolean QTBig_MovieIsStoppable (MovieController theMC)
{
  long   myFlags = 0L;

  MCGetControllerInfo(theMC, &myFlags);

  if ((myFlags & mcInfoIsLooping) ||
        (myFlags & mcInfoMovieIsInteractive))
    return(false);
```

Special Small Dogs

Here are just a few of the photos our customers have sent us of their special dog friends!

Check out all the other special Small Dog photos and send us one of your own at www.smalldog.com!

Small Dog's Special

PowerBook G4/667 Titanium
$3199

- 512MB RAM
- 30gb Hard Drive
- Combo Drive
- Airport card and base station

Purchase from an Apple Specialist!

Small Dog Electronics is proud to have earned Apple's prestigious designation of Apple Specialist. Small Dog Electronics has exceeded all of Apple's stringent requirements for its Specialists and has taken it one step further.

Small Dog sells only the most powerful personal computers in the world—every single one of them an Apple Macintosh! In addition to the rigorous Apple training program, each Small Dog employee has chosen a specific area of expertise to concentrate upon. You can be assured that whomever you talk to at Small Dog, from our sales engineers to our shipping department, you are talking to a trained, enthusiastic, Apple Macintosh Specialist!

Small Dog Electronics is also an Authorized Apple Service Provider Plus. We have certified Apple Technicians that utilize genuine Apple parts for all repairs and provide technical support for our customers based upon their experience in upgrading and supporting thousands of Apple Macintosh computers for our customers!

Talk To an Apple Professional!
Did you know that when you call Small Dog Electronics, you are most likely talking to an Apple Product Professional? Each year, we participate in Apple's on-line courses and seminar training programs to learn as much as we can about the products that we sell and service!

Exclusive Top Dog Membership Treats
For each dollar purchased on-line at the Small Dog web site, you will receive a doggie treat. You can redeem your accumulated treats for Small Dog or Apple apparel and other products. You are automatically enrolled and begin accumulating treats in the Top Dog Club with your first purchase. Remember the more you buy, the more treats for you!

Small Dog Electronics

1673 Main Street
Waitsfield, VT 05673 USA
Phone: 802-496-7171
Online: smalldog.com

Apple Specialist

```
  else
    return(true);
}
```

Now we need to install a time base callback function and respond when the callback function is triggered.

Installing a Time Base Callback Function

To install a time base callback function, we create a *time base callback* and then activate it. We create a time base callback by calling the NewCallBack function, like this:

```
myCallBack = NewCallBack(
        GetMovieTimeBase((**theWindowObject).fMovie),
        callBackAtExtremes);
```

As you can see, the first parameter here is the movie's time base, and the second parameter is a constant that indicates the kind of callback we wish to create. We can set one or both of the two high-order bits in the second parameter to request that our callback function can be called at interrupt time or at deferred task time, using these constants:

```
enum {
  callBackAtInterrupt             = 0x8000,
  callBackAtDeferredTask         = 0x4000
};
```

Setting these bits results in more accurate timing, but the callBackAtInterrupt flag requires that the callback function be interrupt-safe (in particular, that it not cause any memory to be allocated or moved). We don't need extremely accurate timing in returning the movie from fullscreen to normal mode, so we'll leave these bits clear when we call NewCallBack.

To activate a time base callback function, we call the CallMeWhen function, which is declared essentially like this:

```
OSErr CallMeWhen (QTCallBack cb, QTCallBackUPP callBackProc,
        long refCon, long param1, long param2, long param3);
```

Here, cb is the callback we created by calling NewCallBack, and callBackProc is a universal procedure pointer for the callback function. The refCon parameter is a reference constant that is passed to the callback function; as you might have guessed, we'll pass the window object for the fullscreen window as the refCon parameter. The last three parameters to CallMeWhen contain additional information required for the callback and vary depending on the type of the callback. For a callback of type callBackAtExtremes, only the param1 parameter is used; it indicates which movie extreme we want to target. We can use these constants to specify a movie extreme:

```
enum {
  triggerAtStart                  = 0x0001,
  triggerAtStop                  = 0x0002
};
```

In QTBigScreen, we want to keep track of the callback and the callback UPP. Accordingly, we'll add a couple of fields to the application data record (defined in the file ComApplication.h):

```
QTCallBack          fCallBack;
QTCallBackUPP     fCallBackUPP;
```

Finally we can give the definition of the QTBig_InstallCallBack function (Listing 17). It calls NewCallBack and CallMeWhen, and it stores the callback identifier and the callback UPP in the application data record.

Listing 17: Installing a time base callback function

QTBig_InstallCallBack

```
void QTBig_InstallCallBack (WindowObject theWindowObject)
{
  ApplicationDataHdl  myAppData = NULL;
  QTCallBack            myCallBack = NULL;

  if (theWindowObject == NULL)
    return;

  if ((**theWindowObject).fMovie == NULL)
    return;

  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindowObject(theWindowObject);
  if (myAppData == NULL)
    return;

  myCallBack = NewCallBack(
        GetMovieTimeBase((**theWindowObject).fMovie),
        callBackAtExtremes);
  if (myCallBack != NULL) {
    (**myAppData).fCallBack = myCallBack;
    (**myAppData).fCallBackUPP =
        NewQTCallBackUPP(QTBig_FullscreenCallBack);
    CallMeWhen(myCallBack, (**myAppData).fCallBackUPP,
        (long)theWindowObject, triggerAtStop, 0, 0);
  }
}
```

Handling a Time Base Callback

Now our callback is primed and ready to fire when the movie reaches its end. At that point, the function QTBig_FullscreenCallBack is executed. QTBig_FullscreenCallBack is declared like this:

```
PASCAL_RTN void QTBig_FullscreenCallBack
        (QTCallBack theCallBack, long theRefCon);
```

The first parameter is the callback identifier; the second parameter is the reference constant we passed to CallMeWhen, which is of course the window object for the fullscreen window. In general, it's best to keep callback functions short and sweet; the recommended practice is simply to set some flag that is inspected elsewhere in the application. Listing 18 gives our definition of QTBig_FullscreenCallBack.

Listing 18: Handling a time base callback

QTBig_FullscreenCallBack

```
PASCAL_RTN void QTBig_FullscreenCallBack
        (QTCallBack theCallBack, long theRefCon)
{
  WindowObject          myWindowObject =
        (WindowObject)theRefCon;
  ApplicationDataHdl  myAppData = NULL;
  QTCallBack            myCallBack = NULL;

  if (myWindowObject == NULL)
    return;

  myAppData = (ApplicationDataHdl)

QTFrame_GetAppDataFromWindowObject(myWindowObject);
```

```
  if (myAppData == NULL)
    return;

  if (((**myAppData).fCallBack != theCallBack)
    return;

  // mark this window for ending fullscreen mode
  (**myAppData).fEndFullscreenNeeded = true;

  // clean up the callback stuff
  if ((**myAppData).fCallBack != NULL)
    DisposeCallBack((**myAppData).fCallBack);

  if ((**myAppData).fCallBackUPP != NULL)
    DisposeQTCallBackUPP((**myAppData).fCallBackUPP);

  (**myAppData).fCallBack = NULL;
  (**myAppData).fCallBackUPP = NULL;
}
```

First, we cast the theRefCon parameter to be of type WindowObject and make sure that we are given a non-NULL window object. Then we extract the application data associated with that window object and verify that the callback passed to the callback function (theCallBack) is the same as the callback stored in the application data record ((**myAppData).fCallBack). If everything checks out okay, we set the fEndFullscreenNeeded field to true. We'll add some code to the QTApp_HandleEvent function that checks this field and returns a window to normal mode if it is true. Remember that we already go looking for any windows that are marked for returning to normal mode, so we can rework the while loop as shown in Listing 19.

Listing 19: Looking for windows to return to normal mode

QTApp_HandleEvent

```
myWindow = QTFrame_GetFrontMovieWindow();
while (myWindow != NULL) {
  myNextWindow = QTFrame_GetNextMovieWindow(myWindow);

  myAppData = (ApplicationDataHdl)
        QTFrame_GetAppDataFromWindow(myWindow);
  if (myAppData != NULL) {

    if ((**myAppData).fEndFullscreenNeeded) {
      (**myAppData).fEndFullscreenNeeded = false;
      QTBig_StopFullscreen(
        QTFrame_GetWindowObjectFromWindow(myWindow));
    }

    if ((**myAppData).fDestroyWindowNeeded) {
      (**myAppData).fDestroyWindowNeeded = false;
      QTFrame_DestroyMovieWindow(myWindow);
    }
  }

  myWindow = myNextWindow;
}
```

Finally, since we are done with the callback, we call DisposeCallBack to dispose of the callback and DisposeQTCallBackUPP to dispose of the callback UPP. We finish up by clearing out the fields in the application data record that store the callback and callback UPP.

CONCLUSION

Let's quickly recap what we've learned here. A movie can contain a movie user data item of type 'ptv ' that (usually)

Take

C O N t R o L

With JobOrder

Estimating ▪ Scheduling ▪ Job Costing ▪ Traffic ▪ Billing
Reporting ▪ Notifications ▪ Accounting ▪ Calendars

[Discover the Affordable Management Software]

free demo at www.JobOrder.com
toll-free 877.714.2587 phone 607.756.4150

JobOrder™
maximizing business productivity

by Bob Boonstra, Westford, MA

Disk Defragmentation

Occasionally people write me to say that the Challenges in this column are too difficult, that there just isn't enough time to solve the problems in the two plus weeks before the solutions are due. And in looking back, I have to confess that the problems have gotten more difficult over time. I rediscover this trend from time to time, and I've made several attempts to reverse it, only to have the problem difficulty creep up again after a while. Or, as others have written, I take a reasonable problem and put some evil twist into the problem statement, making it impossible. In part, the increased difficulty is because all of the easy problems have already been solved. Like all of the inventions worth inventing have been invented, and discoveries worth discovering have been discovered, and doctoral theses worth writing have been written. Probably not true, it just takes creativity. So this month we'll make another attempt at a simpler problem. In the future, of course, you readers can help by sending in your suggestions, which not only gives you the satisfaction of suggesting a Challenge that can actually be solved in the time available, you earn two invaluable Programmer's Challenge points if we use your suggestion.

This month's Challenge? Defragment a simulated disk drive. For each test case, you are given a disk drive with up to 32767 (2^15-1) blocks. On that drive are a number of files, some allocated contiguously, some allocated in a number of noncontiguous fragments. Your job is to move blocks of data so that the storage for each file is contiguous. It doesn't matter where on the disk the files are located; the free storage remaining on the disk does not need to be contiguous. There is one constraint – the amount of available auxiliary storage is limited, so you cannot remove multiple blocks of data from the disk to make room for others. You can move a contiguous sequence of disk blocks from one location to another in a single operation, as long as they don't overlap.

The file **defrag.in** contains a single line with the number of test cases your program needs to process. The input for each test case is provided in file **defragNN.in**, where NN ranges from 1 to the number of test cases. The first line of this input file contains the number of blocks in the simulated disk to be defragmented. The next line contains the number of data files on that disk. The remainder of the input file is a sequence of lines for each data file. The first line for each data file is the number of fragments in the allocation of the data file on the disk. This is followed by a line for each fragment, containing the disk block where the fragment starts, and then the number of blocks in that fragment. So an input file might look like the following:

```
32767        - number of blocks in the simulated disk
10           - number of files in this test case
1            - number of fragments for the first file
11234,100    - fragment starts at block 11234, uses 100 blocks
3            - number of fragments for the second file
11334,10     - fragment starts at block 11334, uses 10 blocks
11134,100    - fragment starts at block 11134, uses 100 blocks
11344,50     - fragment starts at block 11344, uses 50 blocks
…            - continue for 8 more files
```

Your program should process each input file and output a sequence of block moves to the file **defragNN.out**. Each block move should produce one line of output with 3 numbers:

```
firstBlockToMove,newBlockLocation,numberOfBlockstoMove
```

Finally, your program should produce a **challenge.log** file, with one line per test case containing the integer number of microseconds used by your application to solve that test case, including the time to read the input, find the solution, and produce the output file. The method used to measure execution time may vary based on the development environment you use for your solution, but you should measure time with microsecond precision if possible.

You can improve your chances of winning by incorporating optional features into your solution. For this disk defragmentation problem, you might want to optionally display your solution's progress in defragmenting the disk.

Scoring will be based on minimizing the number of move sequences required to defragment the disk, on minimizing execution time, on a subjective evaluation of additional features, and on the elegance of your code, including the commentary that describes your solution. Your base score will be 100 penalty points for each defragment move sequence. You incur the same number of penalty points for a move of one block as you incur for a single move of multiple contiguous blocks. For each test case, your penalty points are increased by 1% per millisecond of execution time. Your penalty points will be decreased by up to 25% based on any optional features you might incorporate into your solution, and by another 25% based on a subjective evaluation of the elegance of your solution. Since one of the reasons people read the Challenge column is to learn techniques from our Challenge masters, it is important that your code be well documented. Code clarity and commentary will be considered in the evaluation of elegance.

This will be a native PowerPC Challenge, using the development environment of your choice, provided I have or can obtain a copy — email progchallenge@mactech.com to check before you start. You can develop for Mac OS 9 or Mac OS X. Your solution should be a complete Macintosh application, and your submission should provide everything needed to build your application.

A question for you readers, especially those of you that have entered or plan to enter the Challenge: how many of you use Mac OS X regularly? As I write this, Mac OS X has been around for a year or so, and for perhaps half that time in a useable form. I'll confess, I've stuck with Mac OS 9.x until recently, but I've left my most recent machine in its default configuration, booting into Mac OS X 10.1. Although it still annoys the heck out of me on occasion, it is beginning to grow on me. Are we ready to move the Challenge to OS X exclusively? Let me know what you think.

WINNER OF THE JANUARY, 2002 CHALLENGE

The January Challenge required contestants to write a player for TriMineSweeper, a variant on the traditional MineSweeper game. Like the classic MineSweeper, this game requires one to map out an arrangement of cells, discovering which cells are safe and which contain bombs. Moves are made by querying one cell at a time, designating whether that cell is believed to be empty (safe) or to contain a bomb (unsafe). Unlike the classic game, cells in TriMineSweeper are triangular in shape, and each cell has twelve neighbors instead of eight.

Congratulations to **Xan Gregg** for winning the TriMineSweeper Challenge. Xan was significantly more successful than other contestants in solving the TriMineSweeper boards, succeeding in solving eight of the ten test cases I used in the final evaluation. His solution also used less execution time than either of the other top two contestants. His strategy was to make all obviously safe moves, then to evaluate the collective information about the remaining cells to deduce and make other safe moves, and then to "pray" and guess a cell, favoring those cells that have a lower chance of containing a bomb based on what is known from neighboring cells. I suspect the reason Xan's solution was so successful has to do with the way he managed "sets", or collections of unknown cells surrounding a given cell. By combining information

5 times faster

NEW: PRIMEBASE 4.0

Using PrimeBase, you are able to produce Internet and Intranet Applications faster and with the best performance ever.

The development package includes the

- PrimeBase Application Server
- PrimeBase Development Framework
- PrimeBase SQL Database Server

[get developer key free of charge]

download now!
www.primebase.com

Develop on a Mac and deploy without any additional effort on any platform you want.

- Completely cross-platform
- Full-text searching and indexing
- Object-oriented
- Separation of code and layout
- Native connectivity to any database

PRIMEBASE

SNAP Innovation GmbH
Altonaer Poststraße 9a
D-22767 Hamburg / Germany
www.primebase.com
e-mail: info@primebase.com
Fon: ++49 (40) 30629-400
Fax: ++49 (40) 30629-499

from overlapping sets, and by doing so during a "clean up" pass rather than after each move, Xan identified new safe moves and did so efficiently.

The second-place solution by Ernst Munter also used the concept of sets of cells surrounding a given cell, but apparently derived less information from overlapping sets, causing it to have to guess more frequently. Ernst's solution was designed in such a way that it could be adapted relatively easily to other board topologies.

The boards in the evaluation test cases ranged in size from 20x20 to 100x100, with the percentage of cells occupied by bombs ranging from 2.5% to 10%. Half of the test cases had 10% of the cells occupied by bombs, and half had fewer.

The table below lists, for each of the solutions submitted, the total execution time in milliseconds, the number of points earned, equal to the sum of the number of board cells in each game successfully solved, minus one point for each millisecond of execution time. It also lists the code side, data size, and programming language used for each entry. As usual, the number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges prior to this one.

One last note. I received an entry coded in BASIC, but was unfortunately unable to evaluate it because this Challenge required an interface to a test driver written in C. BASIC fans, and users of other development environments, are encouraged to enter this month's Challenge, where solutions are stand-alone applications.

Name	Time (msec)	Points	Code Size	Data Size	Language
Xan Gregg (120)	307.8	46691	8076	4284	C
Ernst Munter (882)	1423.9	19677	4440	314	C++
Tom Saxton (203)	345.3	19154	5716	270	C++
Alan Hart (35)	218.5	16781	3040	134	C
Allen Stenger (82)	273.4	4226	4284	648	C++
Peter Heerboth	10.4	1590	5356	306	C
Douglas O'Brien					Metal BASIC

Top Contestants ...

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 20 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points (24 mo)	Wins (24 mo)	Total Points
1.	Munter, Ernst	275	10	832
2.	Ricken, Willeke	66	3	134
3.	Saxton, Tom	52	1	210
4.	Wihlborg, Claes	49	2	49
5.	Taylor, Jonathan	39	1	63
8.	Gregg, Xan	20	1	140
9.	Mallett, Jeff	20	1	114
10.	Cooper, Tony	20	1	20
11.	Truskier, Peter	20	1	20

... and the Top Contestants Looking for a Recent Win

In order to give some recognition to other participants in the Challenge, we also list the high scores for contestants who have accumulated points without taking first place in a Challenge during the past two years. Listed here are all of those contestants who have accumulated 6 or more points during the past two years.

Rank	Name	Points (24 mo)	Total Points
6.	Boring, Randy	28	144
7.	Sadetsky, Gregory	22	24
12.	Stenger, Allen	19	84
13.	Shearer, Rob	19	62
14.	Schotsman, Jan	16	16
15.	Hart, Alan	14	39
16.	Maurer, Sebastian	11	108
17.	Nepsund, Ronald	10	57
18.	Day, Mark	10	30
19.	Desch, Noah	10	10
20.	Fazekas, Miklos	10	10
21.	Flowers, Sue	10	10
22.	Leshner, Will	7	7
23.	Miller, Mike	7	7

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Introducing

Collaborative Commerce for Small-to-Midsize Businesses and for People, too

For Mac OS X Linux Windows

MyBooks®
MyBooks Professional
Appgen® Custom Suite
Moneydance®

Fourth generation development environment

Variable-length relational database

Commerce modules

Source code

visit www.appgen.com or call 1 800 231 0062

APPGEN®
Collaborative Commerce Platform

Here is Xan's winning TriMineSweeper solution:

Jan02 Solution.c
Copyright © 2002
Xan Gregg

```
/* Solution to TrimineSweeper Programmer's Challenge.
 Solves triangular mine sweeper puzzles.
 Keeps a list of sets of cells. Each set corresponds to the unknown cells surrounding a
 given cell; there is a maximum of 12 cells per set. Each set also knows its total bomb
 count.

 Each cell in the set is represented by its board index, that is, row * boardSize + col.
 Member cells are stored in numerical order to speed searches for cells and set
 comparison.

 When a non-bomb cell is revealed, a set is added with the learned information.

 The play mostly consists of repeatedly making safe moves until there are none.
 "Safe" moves are revealing cells in sets where the bomb count is zero or the bomb
 count is the same as the member count.

 When no safe moves are obvious, the sets are processed to remove known cells and
 eliminate subsets. Removing the known cells as they are discovered turns out to be
 quite a bit slower than doing it in the clean-up pass. For subset elimination, sets that
 are subsets of other sets are reduced so that

  4cells 2bombs (11,12) (11,13) (11,14) (11,15)
  3cells 1bombs (11,12) (11,13) (11,14)

 gets reduced to
  1cells 1bombs (11,15)
  3cells 1bombs (11,12) (11,13) (11,14)

 which produces a safe move for the next move pass. When that fails, the program
will look for certain intersecting sets, such as

  6cells 4bombs (10,12) (11,12) (11,13) (11,14) (11,15) (11,16)
  4cells 1bombs ( 9,12) (11,12) (11,13) (11,14)

 which will produce

  3cells 3bombs (10,12) (11,15) (11,16)
  1cells 0bombs ( 9,12)
  3cells 1bombs (11,12) (11,13) (11,14)

 when the common subset is factored out. Seems to come up in about 1 in 1000
 games, so the value here is debatable.

 When no safe moves or set reductions are possible then the only option is to "pray"
 and reveal a random square. Of course, prayer is required at least for the first few
 moves. Before guessing, sets are compared to determined the safest guess based on
 bombs per member.
*/
#include "Triminesweeper.h"

#include <stdio.h>
#include <stdlib.h>
#include <MacMemory.h>

static const char *gBoard;
static int gBoardSize;
static int gBombsRemaining;
static int gUnknownRemaining;
static MinesweeperMoveProc gMakeMove;
static int gMaxCell;
static Boolean gGameOver;

// arguments for MakeMove function
#define kCheck false
#define kMarkBomb true

// the main data structure: a set of cells
#define kMaxNeighbors 12
typedef struct Set {
    int memberCount;
    int bombCount;
    int members[kMaxNeighbors];
} Set;

// the list of sets, allocated size grows as needed
static int gSetCount;
static int gSetAlloc;
static Handle gSetHandle;
static Set *gSets;

// list of sets that have been touched,
// and thus may need cleaning.
// Fixed size is OK because it's ok if everything can't be recorded
#define kStackSize 1000
static int gTouchedStack[kStackSize];
static int gTouchedCount;

// utilities
#define AsSetIndex(row, col)      (((row) * gBoardSize) +
(col))
#define SetMemberRow(member)      ((member) / gBoardSize)
#define SetMemberCol(member)      ((member) % gBoardSize)
#define CellValue(row,col) (gBoard[(row)*(gBoardSize) +
(col)])
```

InitSets

```
// alloc set list
static void InitSets() {
    gSetAlloc = 1000;      // will grow as needed
    gSetHandle = NewHandle(sizeof(Set) * gSetAlloc);
    HLock(gSetHandle);
    gSets = (Set *) *gSetHandle;
    gSetCount = 0;

    gTouchedCount = 0;
}
```

DisposeSets

```
static void DisposeSets() {
    DisposeHandle(gSetHandle);
}
```

NoteSetAdded

```
// set has been added, so it needs to be checked for simplication
static void NoteSetAdded(int s) {
    if (gTouchedCount < kStackSize)
        gTouchedStack[gTouchedCount++] = s;
}
```

NoteSetReduced

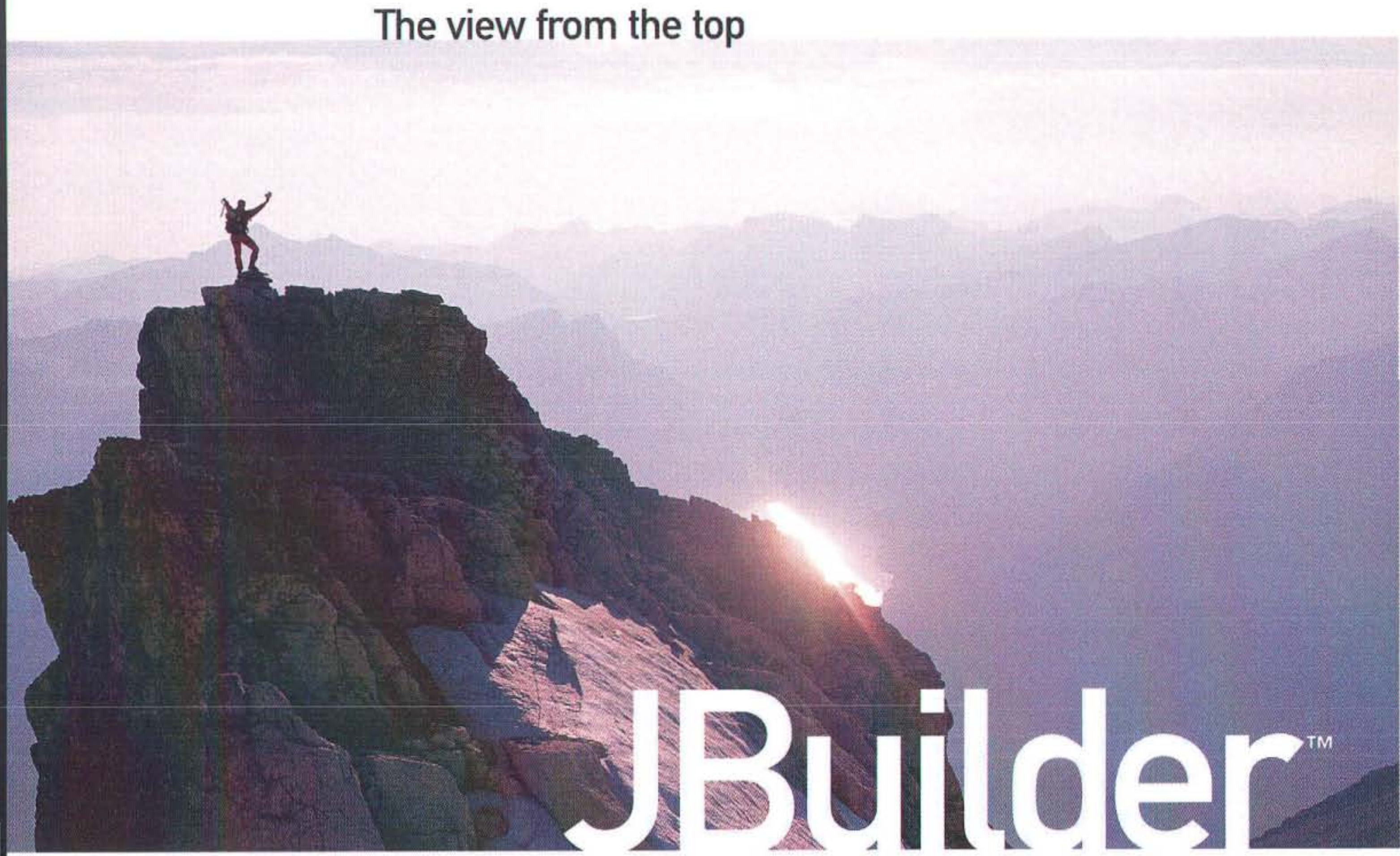
```
// set has been reduced, so it needs to be checked for simplication
static void NoteSetReduced(int s) {
    int i;
    for (i = 0; i < gTouchedCount; i++) {
        if (gTouchedStack[i] == s)
            return;      // already there, so don't add
    }
    if (gTouchedCount < kStackSize)
        gTouchedStack[gTouchedCount++] = s;
}
```

PopTouched

```
// remove a set to be checked
static int PopTouched() {
    if (gTouchedCount > 0)
        return gTouchedStack[-gTouchedCount];
    return 0;
}
```

NoteSetReplaced

```
// set has been replaced by another, so any occurrence
// of 'from' set need to be changed.
static void NoteSetReplaced(int from, int to) {
    int i;
    for (i = 0; i < gTouchedCount; i++) {
        if (gTouchedStack[i] == to) {
            gTouchedStack[i] = gTouchedStack[gTouchedCount -
1];
            gTouchedCount -= 1;
            i -= 1;
        }
        else if (gTouchedStack[i] == from)
            gTouchedStack[i] = to;
    }
}
```



Borland®

The possibilities are limitless with JBuilder, the leading Java™ development solution.

Take EJB development to the next level with an intuitive visual designer. Rapidly develop and deploy J2EE™ e-business applications to multiple application servers including Borland® Enterprise Server, BEA® WebLogic,® IBM® WebSphere,® and iPlanet™ Application Server. Develop and deploy applications on Windows,® Linux,® Solaris,™ and Mac® OS platforms. Efficiently collaborate as a team with support for leading version control systems. Build data-driven Web applications using servlets, JSP,™ and XML.

View code in a dramatic new way with UML code visualization. Experience extreme productivity with refactoring, unit testing, and documentation tools.

See for yourself at http://www.borland.com/new/jb6/5064.html

```
#define kEqual 0
#define kSubset 1
#define kSuperset 2
#define kDisjoint 3
```

CompareSets

```
// compare sets; return equal, subset, superset, or disjoint
static int CompareSets(Set * a, Set * b) {
    int * ap = a->members;
    int * bp = b->members;
    int * aend = ap + a->memberCount;
    int * bend = bp + b->memberCount;
    int am, bm;
    if (a->memberCount == b->memberCount &&
            a->bombCount == b->bombCount) {
        // may be equal
        while (ap != aend && bp != bend) {
            am = *ap++;
            bm = *bp++;
            if (am != bm)
                return kDisjoint;
        }
        if (ap != aend || bp != bend)
            return kDisjoint;
        return kEqual;
    }
    else if (a->memberCount > b->memberCount &&
                a->bombCount >= b->bombCount) {
          // may be superset
        while (ap != aend && bp != bend) {
            am = *ap++;
            bm = *bp++;
            while (am < bm && ap != aend)
                am = *ap++;
            if (am != bm)
                return kDisjoint;
        }
        if (bp != bend)
            return kDisjoint;
        return kSuperset;
    }
    else if (a->memberCount < b->memberCount &&
                a->bombCount <= b->bombCount) {
          // may be subset
        while (ap != aend && bp != bend) {
            am = *ap++;
            bm = *bp++;
            while (am > bm && bp != bend)
                bm = *bp++;
            if (am != bm)
                return kDisjoint;
        }
        if (ap != aend)
            return kDisjoint;
        return kSubset;
    }
    else
        return kDisjoint;
}
```

SubtractSet

```
// a = a - b; b must be a proper subset of a
static void SubtractSet(Set * a, Set * b) {
    int * ap1 = a->members;
    int * ap2 = a->members;
    int * bp = b->members;
    int * aend = ap1 + a->memberCount;
    int * bend = bp + b->memberCount;
    int am, bm;
    while (bp != bend) {
        am = *ap2++;
        bm = *bp++;
        while (am < bm && ap2 != aend) {
            *ap1++ = am;
            am = *ap2++;
        }
    }
    while (ap2 != aend) {
        *ap1++ = *ap2++;
    }
    a->memberCount -= b->memberCount;
    a->bombCount -= b->bombCount;
}
```

SimplifySubsets

```
// Check the modified set against all others.
// Remove if it is equal to one.
// If it has a subset or superset,
// replace the superset with the difference between the sets.
static void SimplifySubsets(Set * modifiedSet) {
    int s;
    Set * set = gSets;
    int answer;
    for (s = 0; s < gSetCount; s++, set ++) {
        if (set == modifiedSet)
            continue;
        answer = CompareSets(set, modifiedSet);
        if (answer == kEqual) {
            gSetCount -= 1;
            *set = gSets[gSetCount];
            NoteSetReplaced(gSetCount, s);
            break;
        }
        else if (answer == kSubset) {
            SubtractSet(modifiedSet, set);
            NoteSetReduced(modifiedSet - gSets);
        }
        else if (answer == kSuperset) {
            SubtractSet(set, modifiedSet);
            NoteSetReduced(set - gSets);
        }
    }
}
```

SimplifySets

```
// simplify all sets that have been touched
static Boolean SimplifySets() {
    if (gTouchedCount != 0) {
        while (gTouchedCount != 0)
            SimplifySubsets(&gSets[PopTouched()]);
        return true;
    }
    return false;
}
```

FindSetMember

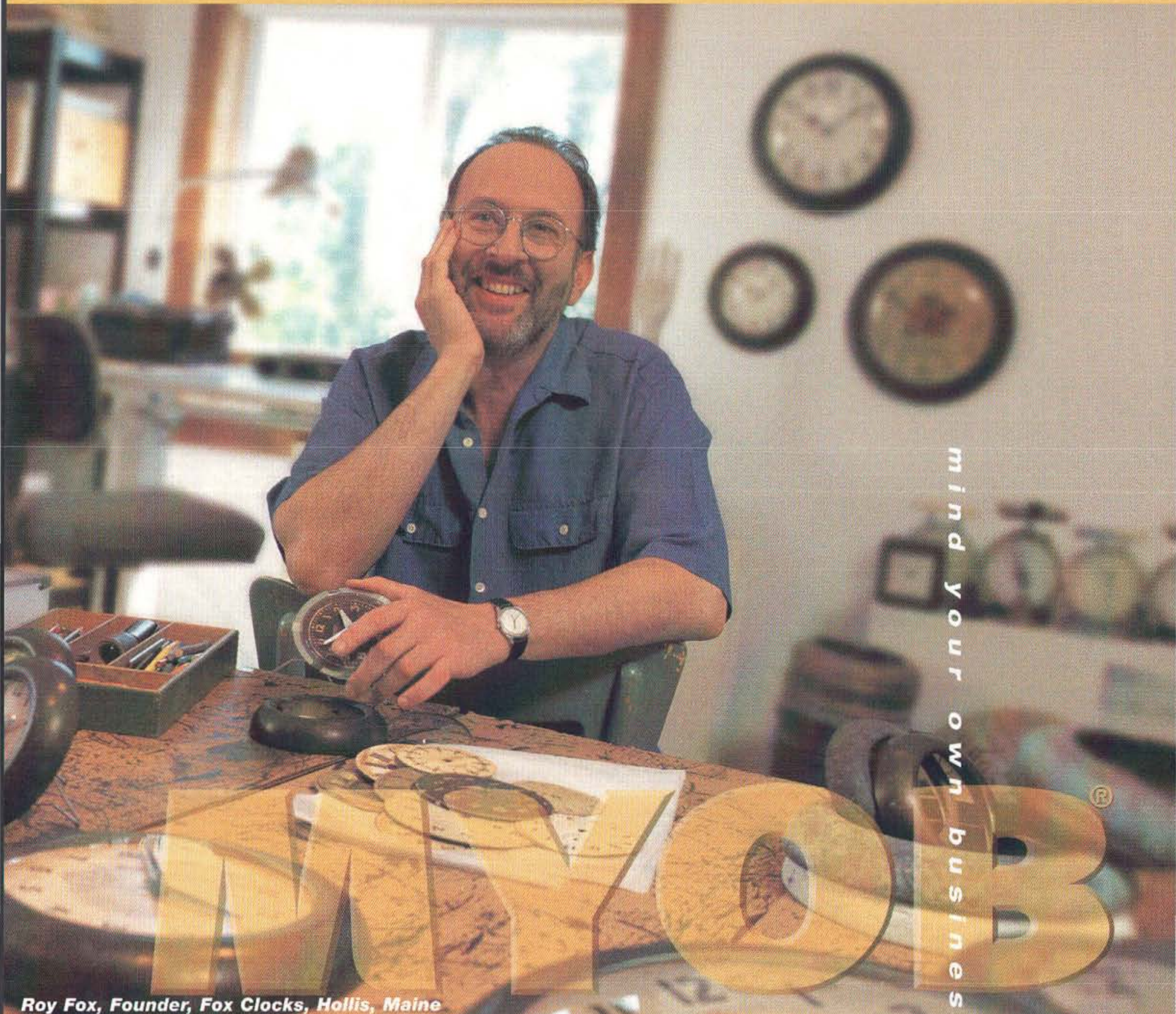
```
// find the given cell in the given set;
// return -1 if not found, otherwise return index
static int FindSetMember(Set * set, int member) {
    int cellCount = set->memberCount;
    int * cells = set->members;
    //fixme — use binary search for larger sets
    int m = 0;
    if (member >= cells[cellCount/2]) {
        m = cellCount/2;
        cells += m;
    }
    for (; m < cellCount; m++, cells++) {
        if (member == *cells)
            return m;
        if (member < *cells)
            break;
    }
    return -1;      // not found
}
```

AllocSet

```
// make sure there is space for another set
static Set * AllocSet() {
    if (gSetCount >= gSetAlloc) {
    // need to allocate more sets
        gSetAlloc = gSetCount * 3 / 2;
        HUnlock(gSetHandle);
        SetHandleSize(gSetHandle, sizeof(Set) * gSetAlloc);
        if (MemError() != noErr) {
            printf("\n\nTROUBLE MISTER - OUT OF
MEMORY\n\n");
            gGameOver = true;
            gSetCount /= 2;
        }
        HLock(gSetHandle);
        gSets = (Set *) *gSetHandle;
    }
    gSetCount += 1;
```

DESIGNED FOR THE WAY YOU WORK

I'd rather create clocks than invoices.
If I wanted to keep books all day, I'd have been an accountant.



Roy Fox, Founder, Fox Clocks, Hollis, Maine
One of 400,000 MYOB software users

MYOB®

Small Business.
Smart Solutions.℠

AccountEdge on the Mac.
Software includes QuickBooks® conversion utility. MYOB also offers conversion service.

800.322.MYOB
myob.com/us

®QuickBooks is a trademark of Intuit, Inc.

MYOB software is the simplest, most powerful, most complete solution for managing my company on the Mac, from the day to day to the bottom line.

Antique frames. Quartz movements. That's my business.
MYOB software works for me.

```
    return &gSets[gSetCount - 1];
}
```

AddNewSet

```
// add a set for the newly revealed cell, being sure
// to add neighbor cells in order of index value
static Set * AddNewSet(int row, int col, int bombCount) {
    Set * set = AllocSet();
    // whether this triangle is up or down pointing:
    int down = ((row+col) & 1);
    int up = 1 - down;
    int r;
    int c;
    int clo, chi;
    int *memp = set->members;
    int index;
    // row above triangle
    if (row != 0) {
        r = row - 1;
        clo = col - 1 - down;
        if (clo < 0)
            clo = 0;
        chi = col + 1 + down;
        if (chi >= gBoardSize)
            chi = gBoardSize - 1;
        index = AsSetIndex(r, clo);
        for (c = clo; c <= chi; c++, index++) {
            int value = gBoard[index];
            if (value == kUnknown)
                *memp++ = index;
            else if (value == kBomb)
                bombCount -= 1;
        }
    }

    // row of this cell
    r = row;
    clo = col - 2;
    if (clo < 0)
        clo = 0;
    chi = col + 2;
    if (chi >= gBoardSize)
        chi = gBoardSize - 1;
    index = AsSetIndex(r, clo);
    for (c = clo; c <= chi; c++, index++) {
        int value = gBoard[index];
        if (value == kUnknown)
            *memp++ = index;
        else if (value == kBomb)
            bombCount -= 1;
    }

    // row below
    if (row != gBoardSize - 1) {
        r = row + 1;
        clo = col - 1 - up;
        if (clo < 0)
            clo = 0;
        chi = col + 1 + up;
        if (chi >= gBoardSize)
            chi = gBoardSize - 1;
        index = AsSetIndex(r, clo);
        for (c = clo; c <= chi; c++, index++) {
            int value = gBoard[index];
            if (value == kUnknown)
                *memp++ = index;
            else if (value == kBomb)
                bombCount -= 1;
        }
    }
    set->memberCount = memp - set->members;
    set->bombCount = bombCount;
    if (set->memberCount == 0) {
        // cancel the add
        gSetCount -= 1;
        set = 0;
    }
    return set;
}
```

CleanSets

```
// remove already revealed cells from all sets;
// helps here that set uses same index that board does,
// so it's efficient to look up cell in board.
static Boolean CleanSets() {
    Boolean cleaned = false;
    int s;
    Set * set = gSets;
    for (s = 0; s < gSetCount; s++, set ++) {
        int * p1 = set->members;
        int * p2 = set->members;
        int * end = p1 + set->memberCount;
        int member;
        int value = kUnknown;
          // move through members until a revealed one is found
        while (p2 != end) {
            member = *p2++;
            value = gBoard[member];
            if (value == kUnknown)
                p1 += 1;
            else {
                set->memberCount -= 1;
                cleaned = true;
                if (value == kBomb)
                    set->bombCount -= 1;
                break;
            }
        }
    // copy members after any revealed members
        while (p2 != end) {
            member = *p2++;
            value = gBoard[member];
            if (value == kUnknown)
                *p1++ = member;
            else {
                set->memberCount -= 1;
                cleaned = true;
                if (value == kBomb)
                    set->bombCount -= 1;
            }
        }
    }
    return cleaned;
}
```

UpdateSets

```
// add set for new cell
static void UpdateSets(int row, int col) {
    char value = CellValue(row, col);
    if (value != kBomb) {      // value == n
    // add a new set for this info
        Set * set = AddNewSet(row, col, value);
        if (set != 0)
            NoteSetAdded(set - gSets);
    }
}
```

MoveAndUpdate

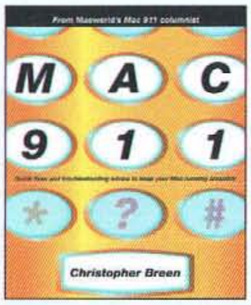
```
// makes a move via callback and adds set if necessary
static void MoveAndUpdate(int row, int col, Boolean
checkOrMark) {
    Boolean youLose = false;
    Boolean youWin = false;
    if (gGameOver)
        return;      // game already over
    gMakeMove(row, col, &gBombsRemaining, checkOrMark,
&youLose,
                                    &youWin);
    gUnknownRemaining -= 1;
    if (youLose || youWin)
        gGameOver = true;      // to escape main loop
    else
        UpdateSets(row, col);
}
```

ExploreSafely

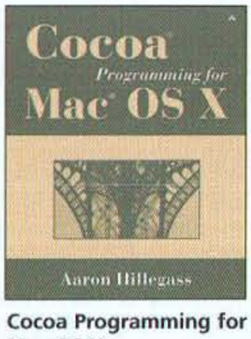
```
// make safe moves. That is, known bombs and known non-bombs.
static Boolean ExploreSafely() {
    Boolean explored = false;
    int s;
    Set * set = gSets;

    for (s = 0; s < gSetCount;) {
    // safe to explore if either no bombs or all bombs
```

Tips, Tricks, and Sage Advice for Your Mac

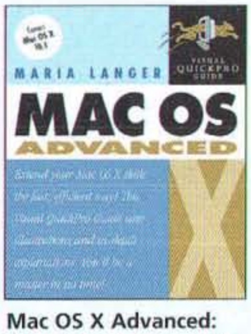
30% off* these Mac titles at BORDERS®



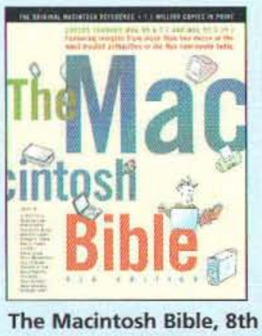
Mac 911
Chris Breen
352 pages • 0-201-77339-2
Retail Price: $29.99
Borders Price: $20.99



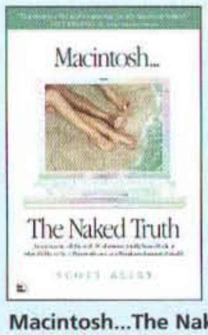
Cocoa Programming for Mac OS X
Aaron Hillegass
416 pages • 0-201-72683-1
Retail Price: $44.99
Borders Price: $31.49



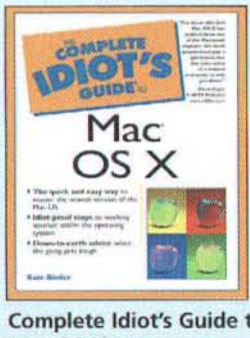
Mac OS X Advanced: Visual QuickPro Guide
Maria Langer
312 pages • 0-201-74577-1
Retail Price: $24.99
Borders Price: $17.49



The Macintosh Bible, 8th Edition
Clifford Colby and Marty Cortinas, Eds.
984 pages • 0-201-70899-X
Retail Price: $34.99
Borders Price: $24.49



Macintosh...The Naked Truth
Scott Kelby
240 pages • 0-7357-1284-0
Retail Price: $19.99
Borders Price: $13.99



Complete Idiot's Guide to Mac OS X
Kate Binder
384 pages • 0-7897-2528-2
Retail Price: $19.99
Borders Price: $13.99

***30% off these select titles 3/24/02 through 4/28/02.**

ALPHA   Peachpit Press   Addison Wesley   New Riders

BORDERS®
BOOKS • MUSIC • MOVIES • CAFE

Made with

REALbasic®

REAL Software and MacTech present the REALbasic Showcase to highlight some of the fantastic solutions created by REALbasic users worldwide. The showcase illustrates the wide range of applications that developers using REALbasic can create. Some benefit any Mac user, and others are more specific. All of them are seriously cool!

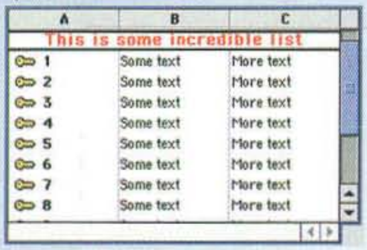
REALbasic is the powerful, easy-to-use tool for creating your own software for Macintosh, Mac OS X, and Windows. It runs natively on Mac OS X as well as earlier versions of the Mac OS. For more information, please visit: **<www.realbasic.com>**.

The Made with REALbasic program is a cooperative effort between REALbasic users and REAL Software, Inc. to promote the products created using REALbasic and the people who create them. For more information about the Made with REALbasic program, please visit: **<www.realbasic.com/realbasic/mwrb/Partners/MwRbProgram.html>**.

Extend REALbasic with Advanced Plugins

Spreadsheet Controls:

We provide a wide range of spreadsheet controls for REALbasic, including multistyled and custom rendering spreadsheet controls.



	A	B	C
	This is some incredible list		
	1	Some text	More text
	2	Some text	More text
	3	Some text	More text
	4	Some text	More text
	5	Some text	More text
	6	Some text	More text
	7	Some text	More text
	8	Some text	More text

- More than 32k of rows.
- Classic, OS X and Win32.
- Accelerated for maximum speed.
- Images in cells.

Cryptography:

We provide data encryption, encoding, compression and hashing for REALbasic.

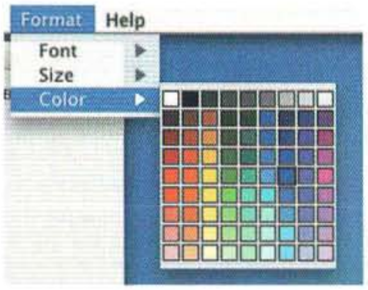
OIIOOIO

Supported Algorithms:

- Encryption:
  - e-CryptIt
  - BlowFish (448 bit)
  - AES (Rijndael) (256 bit)
- Encoding:
  - e-CryptIt Flexible
  - Base 64
  - BinHex
  - MacBinary III
  - AppleSingle / Double
  - UUCoding
- Compression:
  - Zip on Strings and streams (.gz)
- Hashing and Checksums:
  - CRC32 / Adler32
  - MD5 / HMAC_MD5
  - SHA / SHA1 / HMAC_SHA1

Other Plugins:

We have many other plugins for REALbasic, including plugins to do advanced MacOS Toolbox tasks and more custom Controls.



Speed up developement and make more advanced applications by using plugins ! Get free demos at www.einhugur.com

Einhugur Software
sales@einhugur.com
www.einhugur.com

Corona

accounting software

"Easy to set up, easy to use, and excellent support from the developer."

Five stars on VersionTracker.com

- cash entry
- invoicing
- payroll
- reports
- account chart
- address index
- help facility

$49.95

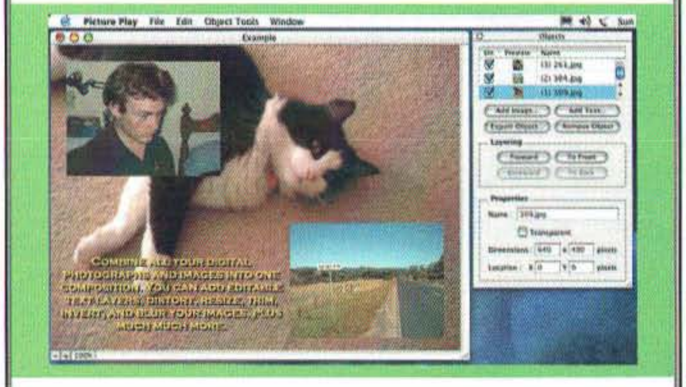
Free 30-day trial
http://homepage.mac.com/idlewild/CoronaUS.hqx

A best friend for business!

p.o. box 3164 • newberg • oregon • 97132
idlewild@mac.com

Picture Play

Create digital image compositions on your Mac, quickly and easily.


Picture Play
File
Edit
Object
Tools
Window
Example
Combine all your digital photographs and images into one composition. You can add editable text layers, distort, resize, trim, invert, and blur your images, plus much much more.

For more information,
email chris@crescendosw.com, or
visit us today at www.crescendosw.com!

db Reports

Printing database driven reports from your REALbasic project has never been easier.

Download your free demo today.

http://abDataTools.com

CREATIVE[BOX]

EAT. SLEEP. DESIGN.

FontViewer

FontViewer is a simple but elegant font viewer that allows you to preview all fonts in your fonts folder. They can be displayed in various sizes and styles with the click of a mouse. FontViewer also features the ability to display your fonts in a slideshow, and view your fonts in almost every type of environment possible (i.e desktop, menus, icons, etc.)

Creative Box URL: http://www.creativebox.net/
Developer e-mail: jason@creativebox.net

piDock

piDock is the must have navigation and launching utility for MacOS. Now you can easily organize, navigate and launch files with a simple sweep of the mouse.
Compatible with OS 8.6 through 10.1, OSX users can now control-option click a folder to navigate it's contents.

http://www.pidog.com • http://www.pidock.com
piDock@pidog.com

iScreensaver Designer

for Macintosh and Windows

Mac designers. . build professional screensavers without touching a Windows machine

the world's only cross-platform screensaver design tool

• build both Macintosh and Windows screensavers with a single click, on whatever system* you are using!

• use any QuickTime movie format : Macromedia Flash, MPEG, Cinepak, MP3, Midi, AVI, DV Video...

• include a hidden movie that can be unlocked with a registration code

• customize and fully-brand both Installers and Screensaver control panels with pictures and text

• screensavers install without DLLs, extensions, or restarts

simple WYSIWYG editor

supports interactive Flash and QuickTime

consistent cross-platform user interface

try before you buy fully functional online downloads

the iScreensaver Designer editing environment

http://iScreensaver.net

email : info @ iScreensaver.net

* supported systems, as of November 2001, include:
Macintosh OS 8.6 to 9.2, Microsoft Windows 95/98/ME, NT4/NT2000/XP

StimpSoft™, Inc.

StimpSoft™, Inc. Makers of insanely mediocre software for cats, dogs, voles, wallabies, mangos, crickets, and people. We put the "L" in "Software".

URL: http://www.stimpsoft.com
Email: john@stimpsoft.com

Whistle Blower

Enterprise server monitor and restart utility
whistleblower.sentman.com

Connect to and validate the response from web servers, cgi scripts and over 23 other types of servers.

Send email, pages and perform unattended restarts via MasterSwitch or PowerKey.

Shifts make sure that the person on call when the server goes down is the one who gets the page.

68k, PPC and Carbon

Web based administration lets you check on and restart your servers from anywhere.

Customize your response to an outage with Apple Script.

email us at whistleblower@sentman.com

Your product can go here!

Are you looking for extremely small out-of-pocket advertising that will reach your target audience?
We will bull you for ads monthly as each issue ships!
Contact our ad sales team today for more details!
Voice: 805/494-9797 x113 • e-Mail: adsales@mactech.com

www.mactech.com

TitleTrack™ JUKEBOX

MAC MUSIC MANAGEMENT FOR SONY® 5 TO 400 DISC CD CHANGERS

Control up to 12 changers (4,800 music CDs)
Control Any Brand Stereo Receiver
Play MP3 and other Sound Files
Plus much, much more!!!

www.titletrack.com info@titletrack.com

REALbasic® For Dummies®

Erick Tejkowski's guide for beginning REALbasic while creating useful and impressive applications is a "must have" in any REALbasic reference library.

This book includes a professional reference that will give you the functionality you need in a hurry.

Go to:
www.devdepot.com/realbasicfordummies
and buy today for over $8 off retail price!

Available While Supplies Last!

DEV DEPOT℠
www.devdepot.com

Unlock the power of OS X!

MONTAGE
Technologies Incorporated

Mac OS X and WebObjects Development and Consulting Services

1-866-245-8485 www.montagetech.com

MacTech News

The below news headlines recently crossed our news desk. For more information on each of these items, check out the MacTech web site <http://www.mactech.com> and search out the below headlines from the home page.

- Bains Software Releases New Versions of MacDICT and SnapperHead
- BitVice MPEG2 video encoder: Professional Decoding, Consumer Price
- Trinfinity Releases Time Track 1.1 For Macintosh And Windows
- Runtime Ships Revolution 1.1.1, New Pricing
- Iceblink Software Releases [info]Batcher Classic
- PC-Mac-Net FileShare v1.5 Streamlines File Transfers
- Now Up-to-Date & Contact 4.2: Offers Backup and New Palm Support
- VideoClix 2.0: Building in Interactive Objects
- Freakin' Magic for FileMaker Pro: Accelerate your 4D WebSTAR
- Sambucus v2.0.0: Time Management Application
- Snard v1.0 Released
- Radio Poster 1.0
- Aladdin Ships Secure Delete- The Digital Document Shredder!
- JTRANSIT v2.0 deploys ColdFusion applications to Mac OS X
- Mac OS X Application allows Scripted Control of FrontBase DataBase
- FileMaker Units Top 7.5 Million
- Markzware Announces Quark 5.0 & InDesign 2.0 Support
- eSellerate Supports REALbasic Development For Windows
- LassoSites.com
- St. Clair Software Updates Default Folder X
- Dragon Burn: New CD-Recording Software for Mac
- WestCiv releases Layout Master 1.1 HTML + CSS web page layout
- FILTERiT 4.1.1
- MACWEB.COM - Palo Alto Mac Server Colocation Community
- Cross-Platform HELP Engine for Your REALbasic Apps
- Precision Plugin 1.5 for REALbasic, now with x86 support
- Znippetizer-X version 1.4
- MassTransit Gains Improved Encryption and Security Capabilities
- Presenta Ltd. announces iGetter 1.8
- Colourfull Creations Releases Calendar Class for REALbasic
- eOrdering Complete 1.0 Online Store and Shopping cart creator
- OpenLink Releases Universal Server for Web Services
- Berkeley Releases 802.11b DSSS Analysis System
- Whistle Blower 3.0 for Mac OS X
- PrefsOverload 4.1.2 for Mac OS and Mac OS X Released
- MaxBulk Mailer v2.6
- SMTP Deux Professional v1.1.0 Released by Deep Sky Technologies
- ActiveDeveloper in v2.10 for Mac OS X Server
- Autolycus Announces Mac OS X Compatibility
- Autolycus Offers PC SpinImage DV Pro
- DeviceWatcher for InterMapper Improves Network Services Stability
- Runtime Labs Announces Release Of MacSQL 2.2
- The Omni Group Ships OmniGraffle 2.0
- Troi Dialog Plug-in 3.1 for FileMaker Pro 5.5
- Macworld Conference & Expo Adds New Conferences For New York
- StickyBrain 2
- Waves In Motion Announces Dragon Web Surveys / Quizzes 6.5
- SCRIPTit 1.0 Released (Mac and Win)
- CDFinder 3.6
- Advenio announces SQLGrinder 1.2 with AppleScript support
- ProVUE Ships Panorama iPod Organizer
- illumineX updates Mac OS X games
- Java-Based BlackBerry Handheld With Integrated Phone
- JAW software ships MPEG2Splitter 1.3
- Global Graphics Launches Jaws PDF Server Solution
- Ben Bird Announces BTV Pro 5.4: Update To Macintosh Video Software

Belkin USB VideoBus II for Mac

only **$79.95**

Dr. Bott VGA Extractor for ADC

only **$32.95**

SMARTLINK USB File Transfer Cable

only **$39.95**

2nd Wave Firewire to SCSI

only **$78.95**

DEV DEPOT℠

DevDepot your source for the hottest new Tech Tools!

www.devdepot.com

Voice: 877-DEPOT-NOW (877-337-6866)
Outside US/Canada: 805/494-9797
E-mail: orders@devdepot.com

```
        if (set->bombCount == 0 ||
                set->bombCount == set->memberCount) {
            int m;
            Boolean checkOrMark = set->bombCount != 0;
                            // false means check
            explored = true;
            for (m = 0; m < set->memberCount; m++) {
                int member = set->members[m];
                MoveAndUpdate(SetMemberRow(member),
                        SetMemberCol(member), checkOrMark);
            }
            // remove this set
            gSetCount -= 1;
            *set = gSets[gSetCount];
            NoteSetReplaced(gSetCount, s);
        }
        else {
            s += 1;
            set += 1;
        }
    }
    return explored;
}
```

FindInOtherSet

```
// Look for the given member in all sets except
// for the given one.
static int FindInOtherSet(int knownMember, Set * knownSet) {
    int s;
    Set * set = gSets;
    for (s = 0; s < gSetCount; s++, set ++) {
        int m = FindSetMember(set, knownMember);
        if (m >= 0 && set != knownSet)
            return m;
    }
    return -1;
}
```

PrayInSet

```
// Pick a random cell from the given set and reveal it.
// set == 0 means pick from enire board of unknowns.
static void PrayInSet(Set * set, Boolean checkOrMark) {
    int row;
    int col;
    if (set == 0) {
        if (gUnknownRemaining < gMaxCell / 8) {
            // pick first unknown
            int i;
            for (i = 0; i < gMaxCell; i++) {
                if (gBoard[i] == kUnknown) {
                        // found an unknown cell, so test it
                    row = i / gBoardSize;
                    col = i % gBoardSize;
                    break;
                }
            }
        }
        else {
            // pick one at random until an unknown is found
            while (true) {
                int cell = rand() % gMaxCell;
                if (gBoard[cell] == kUnknown) {
                    // found an unknown cell, so test it
                    row = cell / gBoardSize;
                    col = cell % gBoardSize;
                    break;
                }
            }
        }
    }
    else {
            // prefer last member — quickest to remove
        int m = set->memberCount - 1;
            // also prefer cell that is not within another set,
            // this that set likely has worse odds.
            // this is costly, but it does make it more likely
            // of finding a correct solution.
        while (m > 0 && FindInOtherSet(set-
>members[m],set)>=0)
            m -= 1;

        row = SetMemberRow(set->members[m]);
        col = SetMemberCol(set->members[m]);
    }
    MoveAndUpdate(row, col, checkOrMark);
}
```

Pray

```
#define kOddsScale 4096

// Find the set with the best oods for guessing and make the guess.
static void Pray() {
    Set * praySet = 0;
    int prayOdds = kOddsScale *
                    (gUnknownRemaining - gBombsRemaining)
                        / gUnknownRemaining;
    Boolean prayCheckOrMark = kCheck;
    int s;
    if (prayOdds < 0.5) {
        prayOdds = kOddsScale * gBombsRemaining /
                            gUnknownRemaining;
        prayCheckOrMark = kMarkBomb;
    }
    if (gUnknownRemaining * 2 < gMaxCell)
        prayOdds = 0;    // too few unknown to be accurate
    for (s = 0; s < gSetCount; s++) {
        Set * set = &gSets[s];
        int cells = set->memberCount;
        int bombs = set->bombCount;
        int odds = kOddsScale * (cells - bombs) / cells;
        Boolean checkOrMark = kCheck;
        if (odds < 0.5) {
            odds = kOddsScale * bombs / cells;
            checkOrMark = kMarkBomb;
        }
        if (odds >= prayOdds) {
            praySet = set;
            prayOdds = odds;
            prayCheckOrMark = checkOrMark;
        }
    }
    PrayInSet(praySet, prayCheckOrMark);
}
```

IntersetSets

```
// returns number of common members
// assumes each set has at least two members
static int IntersetSets(Set * a, Set * b, Set * common) {
    int * ap = a->members;
    int * bp = b->members;
    int * cp = common->members;
    int * aend = ap + a->memberCount;
    int * bend = bp + b->memberCount;
    int am, bm;
    int n = -1;
    if (*ap > *(bend-1) || *bp > *(aend-1))
        return 0;
    am = 0;
    bm = 0;
    while (true) {
        if (am == bm) {
            if (n >= 0)
                *cp++ = am;
            n += 1;
            if (ap == aend)
                break;
            am = *ap++;
            if (bp == bend)
                break;
            bm = *bp++;
        }
        else if (am < bm) {
            if (ap == aend)
                break;
            am = *ap++;
        }
        else if (am > bm) {
            if (bp == bend)
                break;
            bm = *bp++;
        }
    }
    common->memberCount = n;
    common->bombCount = 1;
    return n;
}
```

RemoveIntersection

```
// Remove the common subset from A and B, add it as a new set
static void RemoveIntersection(Set *a, Set *b, Set * common)
{
    Set * set = AllocSet();
    *set = *common;
    NoteSetAdded(set - gSets);
    SubtractSet(a, common);
    NoteSetReduced(a - gSets);
    SubtractSet(b, common);
    NoteSetReduced(b - gSets);
}
```

EliminateIntersections

```
// eliminate intersections that fit a specific pattern
// consisting one dense set and one sparse set
static Boolean EliminateIntersections() {
    int s;
    Set * set = gSets;
    Set common;
    if (gUnknownRemaining > gMaxCell - 10)
        return false;       // don't bother early on in game
    for (s = 0; s < gSetCount; s++, set ++) {
        if (set->bombCount >= 3
                && set->memberCount - set->bombCount <= 2)
{
                // found a crowded set (eg, 4 bombs out of 6)
                // now look for an overlapping sparse one
            int needed = set->memberCount - set->bombCount +
1;
            int sb;
            Set * setb = gSets;
            for (sb = 0; sb < gSetCount; sb++, setb ++) {
                if (setb->bombCount == 1
                        && setb->memberCount > needed) {
                    int actual =
IntersetSets(set,setb,&common);
                    if (actual >= needed) {
                            // we don't have to pray after all
                RemoveIntersection(set, setb, &common);
                            return true;
                        }
                    }
                }
            }
        }
    }
    return false;
}
```

PlayMinesweeper

```
// explore, clean, simplify, pray
void PlayMinesweeper (
    int boardSize,
    /* number of rows/columns in board */
    int numberOfBombs,
    /* number of bombs in board */
    const char *board,
    /* board[row*boardSize + col] is board element (row,col)
*/
    MinesweeperMoveProc MakeMove
    /* procedure for reporting moves */
    /* MakeMove updates elements of board */
) {
    gBoardSize = boardSize;
    gBoard = board;
    gBombsRemaining = numberOfBombs;
    gUnknownRemaining = boardSize * boardSize;
    gMakeMove = MakeMove;
    gMaxCell = boardSize * boardSize;
    gGameOver = false;
    InitSets();

    while (!gGameOver) {
        if (!ExploreSafely()
                && !CleanSets()
                && !SimplifySets()
                && !EliminateIntersections()
                ) {
            Pray();
        }
    }
    DisposeSets();
}
```

Mac-zimum Mac-Talk

Your Digital Life...

MacGaming...

TechTips...

Interviews...

MacBasics...

MacMediaPro...

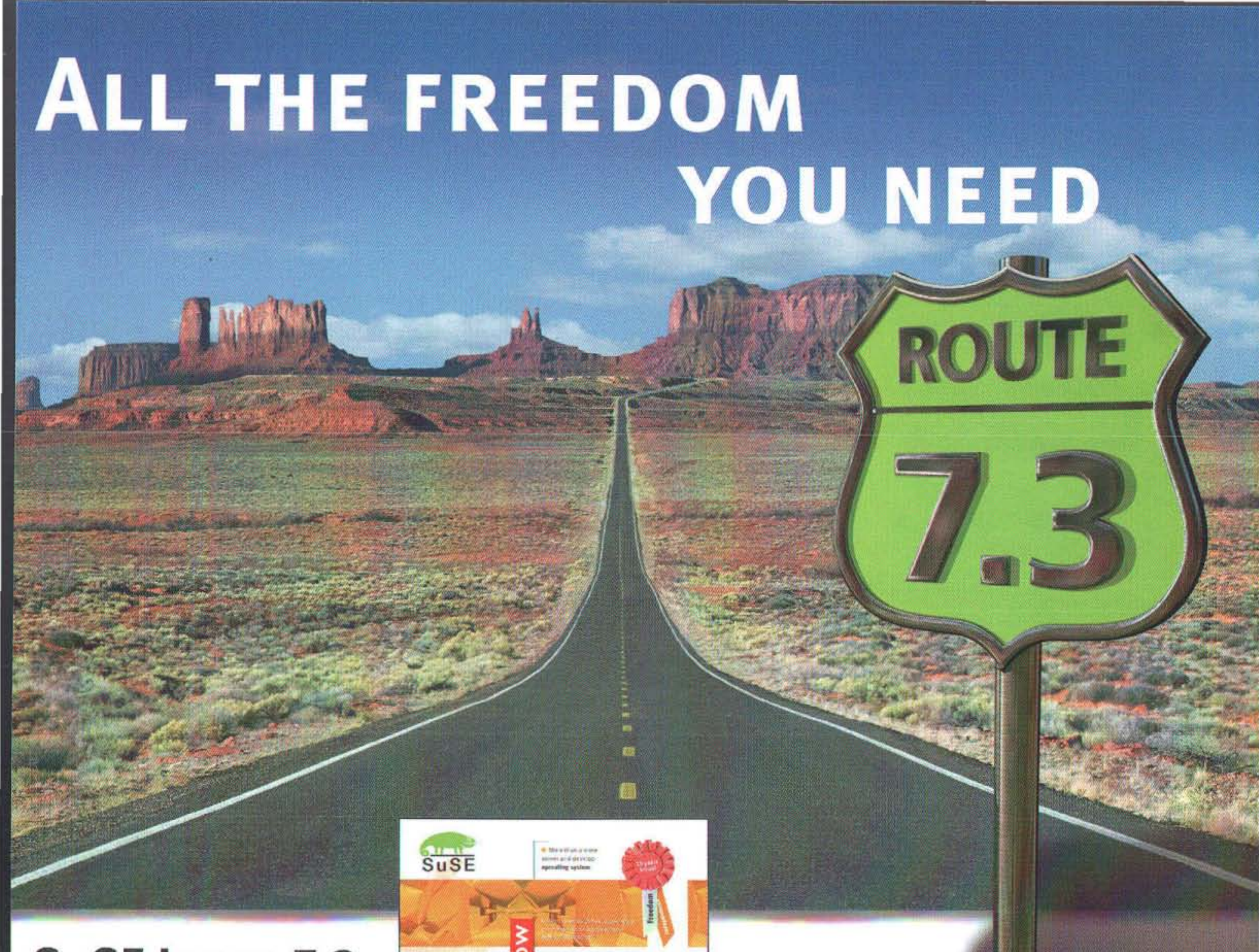
MAC LIVE SHOW

Live WED 9–11PM ET @ macshowlive.com

ADVERTISER/ PRODUCT LIST

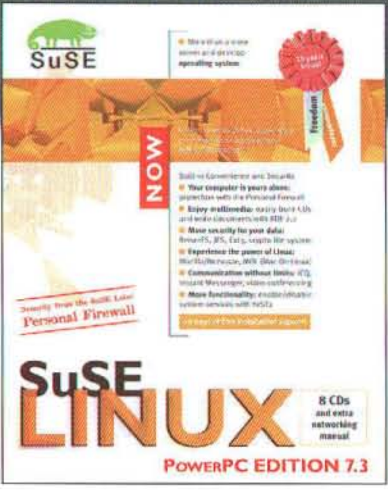
List of Advertisers

List of Products



SuSE Linux 7.3

- Secure
- Stable
- Superior



**SuSE Linux 7.3
PowerPC EDITION:**
$ 79,95

Yes, please send me the power-pack ...

... because Linux users enjoy the power and stability of SuSE Linux.
Don't compromise.

SuSE Linux 7.3 – no limits.

- Full integration with MOL (Mac-On-Linux)
- Scanning made easy with the KDE tool Kooka NEW
- Protect your data with Soft-RAID NEW
- Great sound with ALSA
- Surf safely with the Personal Firewall
- Professional installation support by phone, fax and e-mail
- etc.

Please bill the following address:

Company
Last name/Name
Phone
Street
Zip-code/Postal-code

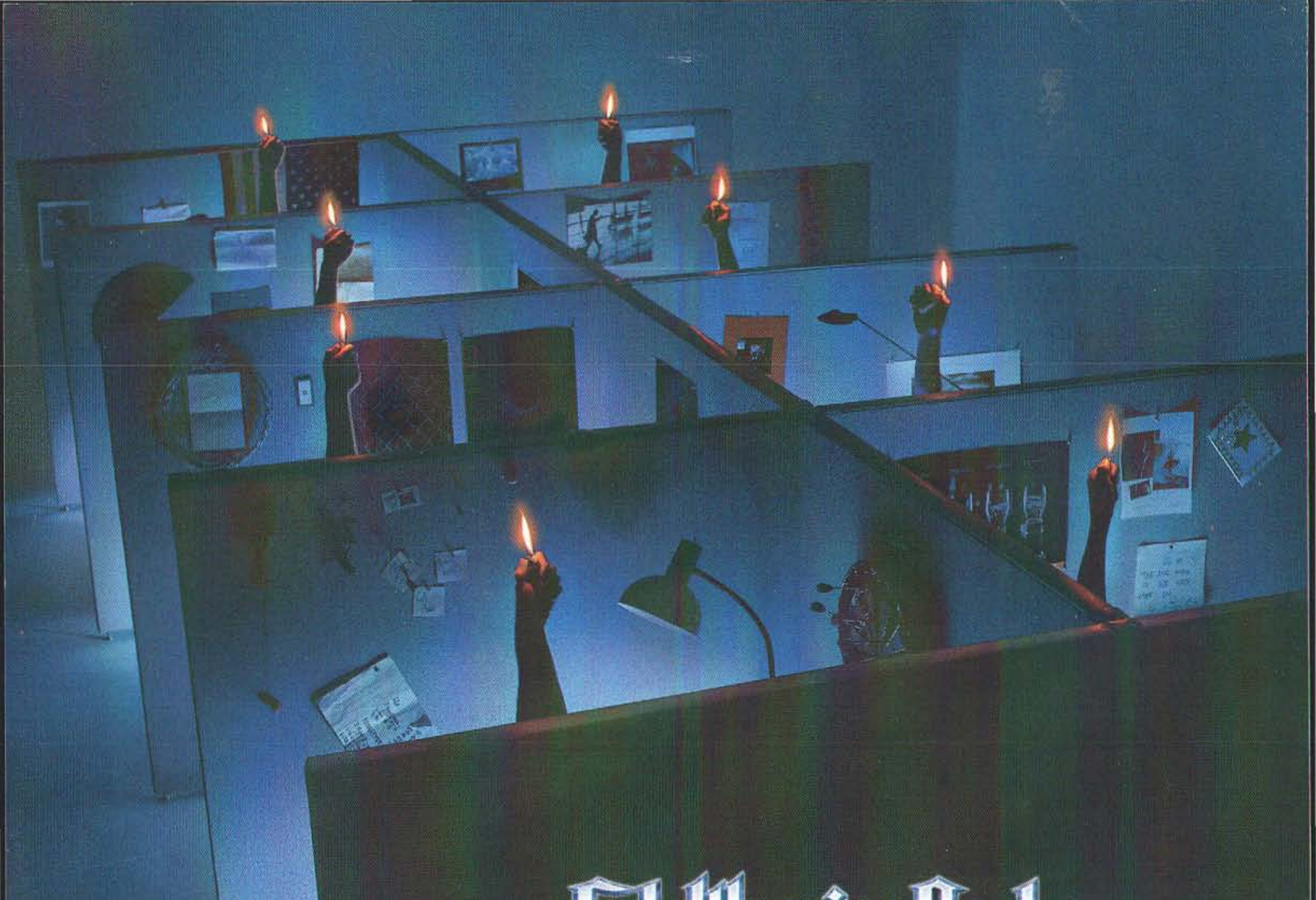
☐ SuSE Linux 7.3 PowerPC EDITION
$ 79.95

SuSE Inc. – The Linux Experts
580 Second St.
Oakland · CA 94607

Order Online Today! http://shop.suse.com/mt

info@suse.com (888) 875-4689 (510) 628-3381

SuSE



CodeWarrior Rocks

Other software companies have customers. We've got fans. In fact, Metrowerks is the leading provider of development tools for the Macintosh® community. With Metrowerks CodeWarrior™ tools, you can build powerful applications with our popular PowerPlant™ framework, which uses industry-standard C++. Best of all, you can create a single application that runs on both Classic Mac® OS and Mac OS X. So choose CodeWarrior tools and get ready to rock.

Register now to win an iPod™ MP3 player at www.metrowerks.com/mactech

CodeWarrior™
Development Tools for Mac® OS

metrowerks
Software Starts Here